



4.9.2-lab - -integrate-a-rest-api-in-a-python-application es-XL Act

Administración de Redes (Instituto Tecnológico de León)



Scan to open on Studocu

Laboratorio - Integrar una API REST en una aplicación Python

Objetivos

Parte 1: Iniciar la Máquina Virtual (VM) DEVASC

Parte 2: Demuestre la aplicación Graphhopper

Directions Parte 3: Obtenga una clave de API

Parte 4: Crear la aplicación de codificación geográfica de Graphhopper

Parte 5: Crear la aplicación de routing de Graphhopper

Parte 6: Probar la funcionalidad completa de la aplicación

Aspectos básicos/Situación

En este laboratorio, creará una aplicación en Visual Studio Code (VS Code) que recupere datos JSON de la API Graphhopper Directions, analice los datos y les dé formato para que se envíen al usuario. Utilizará la solicitud GET Route de la API Graphhopper Directions. Consulta la documentación de la API Directions aquí:

<https://docs.graphhopper.com/>

Nota: Si el enlace anterior ya no funciona, busca «Documentación de la API de Graphhopper Directions».

Recursos requeridos.

- Una computadora con el sistema operativo de su elección
- Virtual Box o VMWare.
- Máquina virtual DEVASC.

Instrucciones

Parte 1: Ejecute la máquina virtual (Virtual Machine) de DEVASC.

Si aún no ha completado el **Laboratorio: Instale el Entorno de Laboratorio de Máquina Virtual**, hágalo ahora. Si ya ha completado ese laboratorio, ejecute la máquina virtual (Virtual Machine) de DEVASC.

Parte 2: Demuestre la aplicación Graphhopper Directions

Su instructor puede hacer una demostración de la aplicación Graphhopper Directions y mostrarle el script utilizado para crearla. Sin embargo, creará este script paso a paso en este laboratorio.

La aplicación solicita un punto de partida y un destino y el modo de transporte. A continuación, solicita datos JSON de las API de geocodificación y enrutamiento de Graphhopper, los analiza y muestra información útil.

```
+++++
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**

Nota: Para ver el JSON para la salida anterior, puede copiar la URL en una pestaña del navegador. However, you will need to replace **your_api_key** with the Graphhopper API key you obtain in Part 3.

Parte 3: Obtener una clave de API

Antes de crear la aplicación, debes completar los siguientes pasos para obtener una clave de API de Graphhopper.

- Vaya a: <https://www.graphhopper.com/>.
- Haz clic en **Sign Up** en la parte superior de la página.
- Rellene el formulario para crear una nueva cuenta. Para **Company**, escriba **Cisco Networking Academy Student**.

- d. Después de verificar su cuenta de correo electrónico e iniciar sesión en Graphhopper, haga clic en **Claves de API** en la parte superior de la página. Haga clic en **Add API Key (Agregar clave de API)** para agregar una nueva clave de API. Proporcione una descripción de la clave de API si lo desea. Haga clic en **Add Key (Agregar clave)** para crear la nueva clave de API.

Nota: Consulte la carpeta de SPAM para ver el correo electrónico de confirmación de Graphhopper si no ha recibido el correo electrónico de confirmación.

- e. Copia tu **clave de API** en un archivo de texto para usarla en el futuro. Esta será la clave que usted usará para el resto de este laboratorio.

Nota: Graphhopper puede cambiar el proceso para obtener una clave. Si los pasos anteriores ya no son válidos, busca en Internet «pasos para generar la clave api de graphhopper»

Parte 4: Crear la aplicación de codificación geográfica de Graphhopper

En esta parte, creará un script de Python para enviar una solicitud de URL a la API de geocodificación de Graphhopper. A continuación, probarás tus llamadas a la API. A lo largo del resto de este laboratorio, creará el script en partes, guardando el archivo con un nombre nuevo cada vez. Esto ayudará con el aprendizaje de las piezas de la aplicación, así como le proporcionará una serie de scripts a los que puede volver si tiene algún problema en la versión actual de su aplicación.

Paso 1: Cree un nuevo archivo en Visual Studio Code (VS Code).

Puede usar cualquier herramienta que desee introducir en los comandos Python y ejecutar el código Python. Sin embargo, este laboratorio demostrará la creación de la aplicación en VS Code.

- a. Cree una nueva carpeta en el directorio `~ / labs / devnet -src`.
- b. Abrir **VS(Visual Studio) Code(Codigo)**. There is a shortcut on the **Desktop**, for your convenience.
- c. Seleccione **File > Abrir Folder...**
- d. Navegue al directorio `~/labs/devnet-src/graphhopper` y haga clic en **OK**.
- e. Seleccione **File > New File**.
- f. Seleccione **File > Save as...** y asigne al archivo el nombre `graphhopper_parse-json_1.py` y haga clic en **Save**.

Paso 2: Importación de módulos para la aplicación.

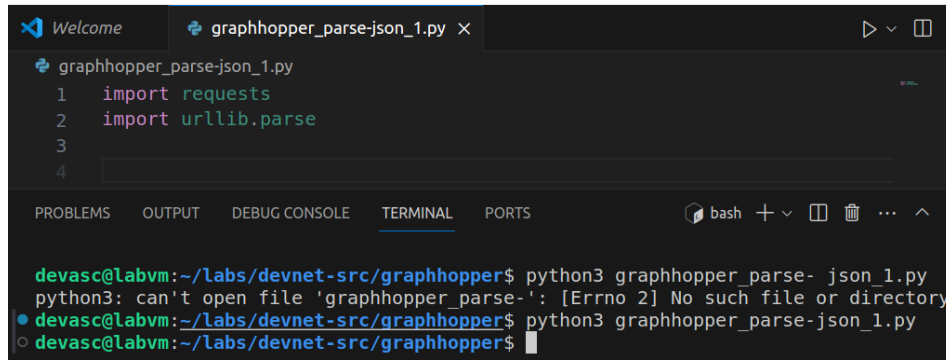
Para comenzar tu script para analizar los datos de JSON, necesitarás importar dos módulos de la biblioteca de Python: **requests** y **urllib.parse**. El módulo **requests** proporciona funciones para recuperar datos JSON de una URL. El módulo **urllib.parse** proporciona una variedad de funciones que le permitirán analizar y manipular los datos JSON que recibe de una solicitud a una URL.

- a. Agregue las siguientes instrucciones de importación en la parte superior del script.

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

- b. Seleccione **Terminal > New Terminal** para abrir una terminal dentro de VS Code.
- c. Guardar y ejecutar el script. No debería recibir errores. Debe guardar y ejecutar los scripts a menudo para probar la funcionalidad del código.

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

```
graphhopper_parse-json_1.py
1 import requests
2 import urllib.parse
3
4

devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse- json_1.py
python3: can't open file 'graphhopper_parse-': [Errno 2] No such file or directory
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_1.py
devasc@labvm:~/labs/devnet-src/graphhopper$
```

Paso 3: Cree la URL para la solicitud a la API de codificación geográfica de Graphhopper.

El primer paso para crear su solicitud de API es construir la URL que su aplicación utilizará para realizar la llamada. Inicialmente, la URL será la combinación de las siguientes variables:

- **geo_url** : la URL de codificación geográfica para solicitar información de longitud y latitud de una ubicación
- **route_url** : la URL de routing para solicitar información de routing desde el punto de origen hasta el destino
- **loc1** - el parámetro para especificar el punto de origen
- **loc2** - el parámetro para especificar el destino
- **key** - la clave API de Graphhopper que recuperaste del sitio web de Graphhopper

- a. En el siguiente código, reemplaza **your_api_key** por la clave de API que copiaste de tu cuenta de Graphhopper.

```
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" # Reemplazar con su clave de API de Graphhopper
```

- b. Combine las tres variables **geocode_url**, **loc1**, y **key** para formatear las URL solicitadas para la longitud y la latitud del origen. Use the **urlencode** method to properly format the address value. Esta función construye la parte de parámetros de la URL y convierte posibles caracteres especiales en el valor de la dirección en caracteres aceptables (por ejemplo, espacio en «+» y una coma en «%2C»). Para la variable de **url**, usaremos la variable de ubicación de origen **loc1** y el límite a un conjunto de coordenadas de latitud y longitud, y la clave de la API de Graphhopper. Reemplazará la variable **loc1** cuando cree una función de geocodificación más adelante en la práctica de laboratorio.

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

- c. Crea variables para guardar la respuesta de la URL solicitada e imprimir los datos JSON devueltos. La variable **replydata** contiene los datos de la respuesta de la URL de Graphhopper. La variable **json_data** contiene una representación en el diccionario de Python de la respuesta json del método **get** del módulo **requests**. El **requests.get** realizará la llamada de API a la API de geocodificación de Graphhopper. La declaración **print** se utilizará temporalmente para comprobar los datos devueltos. Reemplazará esta declaración de impresión con opciones de visualización más sofisticadas más adelante en el laboratorio.

```
replydata = request.get(url)
json_data = replydata.json()
json_status =
replydata.status_code imprimir
(json_data)
```

- d. Su código final deberá verse así, pero con un valor diferente para la clave.

```
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" ## Reemplazar con su clave de API

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)

```

Paso 4: Pruebe la solicitud de URL.

- Guarde y ejecute el script **graphhopper_parse-json_1.py** y compruebe que funciona.
- Solucione el problema de su código, si es necesario. Aunque su salida puede ser ligeramente diferente, debería obtener una respuesta JSON similar a la siguiente. Observe que la salida es un diccionario con dos pares clave/valor. El valor de la clave **hits** es una lista que incluye diccionarios y listas adicionales. La clave **point** incluye el par clave/valor **lat** y **lng** que utilizará más adelante en el laboratorio.

```

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$

```

```

• 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, -76.90
9366, 38.995968], 'name': 'Washington', 'country': 'United States', 'countrycode': 'US', 'state': 'Dist
rict of Columbia', 'osm_id': 5396194, 'osm_type': 'R', 'osm_key': 'place', 'osm_value': 'city'}], 'loca
le': 'default'}
○ devasc@labvm:~/labs/devnet-src/graphhopper$

```

- Cambie la variable **loc1**. Vuelva a ejecutar el script para obtener resultados diferentes. Para garantizar los resultados que desea, lo mejor es incluir tanto la ciudad como el estado para las ciudades de los Estados Unidos. Al referirse a ciudades de otros países, normalmente puede usar el nombre en inglés de la ciudad y el país o el nombre nativo. Por ejemplo:

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

O

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

```

• devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_1.py
{'hits': [{'point': {'lat': 41.8933203, 'lng': 12.4829321}, 'extent': [12.2344669, 41.6556417, 12.85576
03, 42.1410285], 'name': 'Roma', 'country': 'Italia', 'countrycode': 'IT', 'state': 'Lazio', 'osm_id':
41485, 'osm_type': 'R', 'osm_key': 'place', 'osm_value': 'city'}], 'locale': 'default'}
○ devasc@labvm:~/labs/devnet-src/graphhopper$

```

Paso 5: Imprima la URL y compruebe el estado de la solicitud JSON.

Ahora que sabe que la solicitud JSON está funcionando, puede agregar algo más de funcionalidad a la aplicación.

- Guarde el script como **graphhopper_parse-json_2.py**.
- Elimine la sentencia **print (json_data)**, ya que ya no necesitará comprobar que la solicitud tiene el formato correcto.
- Agregue las declaraciones a continuación, que hará lo siguiente:

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)

```

- Utilice la propiedad **status_code** para determinar el estado HTTP respondido.

Laboratorio - Integrar una API REST en una aplicación Python

- Inicie un bucle **if** que compruebe si la llamada se ha realizado correctamente, lo que se indica con un valor devuelto de 200. Si la solicitud es correcta, agregue una declaración de impresión para mostrar la URL de solicitud de codificación geográfica construida. **\n** agrega una línea en blanco antes de mostrar la URL.

Paso 6: Mostrar los datos JSON en JSONView.

- a. Pega la URL de Washington, D.C., en el campo de dirección del navegador Chromium.
- b. Los resultados solo tienen dos diccionarios raíz: coincidencias y configuración regional. Amplíe los **resultados** e investigue los datos enriquecidos. Continúe expandiendo el resultado según sea necesario hasta que se muestren las coordenadas de latitud y longitud del origen o el destino.

```
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=981e87e8-6a0d-42eb-bbe5-1bd9111c6225

{
  - 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"
}
```

```
{
  - 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"
}
```

Paso 7: Cree la función de codificación geográfica.

Hasta ahora, ha creado para la URL la latitud y la longitud de la ubicación de inicio y ha recibido la información de la solicitud de la API. ¿Y la información del destino? Puede reutilizar el mismo código para la segunda ubicación mediante una función.

En este paso, creará una función que acepta las entradas de ubicación y devuelve el valor de **status_code** de la URL de la API de geocodificación, la longitud y la latitud de la ubicación e información adicional de Graphhopper con respecto a la ubicación.

- Este ejemplo utiliza los siguientes parámetros como entrada de ubicación en la función de codificación geográfica.

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

- Elimine la declaración **geocode_url** porque será parte de la función de geocodificación.
- Copie y pegue la nueva función y reemplace todas las declaraciones debajo de su definición de clave de API. Se crea una URL para solicitar los datos JSON mediante la ubicación y los argumentos clave enviados a la función. La función devuelve los valores almacenados en las variables **json_status**, **lat** y **lng**, si la llamada de la API a la API de codificación geográfica fue correcta. De lo contrario, la función devolverá el error **json_status** y valores nulos para las variables de **lat** y **lng**.

```
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
```

Paso 8: Pruebe su nueva función de codificación geográfica.

- a. Para usar la función, creará dos variables para llamar a la función de geocodificación y almacenar los valores devueltos por la función. Copie y pegue las siguientes líneas al final del guion. La función de geocodificación toma los valores de las variables de entrada para `loc1` y `key`, y devuelve la tupla con los valores de las variables `json_status`, `lat` y `lng`. La función se vuelve a llamar para `loc2`.

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

- b. Su código final deberá verse así, pero con un valor diferente para la clave.

```
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) imprimir (destino)
```

- c. Guarda y ejecuta el script **graphhopper_parse-json_2.py** y comprueba que funciona. Solucione el problema de su código, si fuera necesario Usted debería de obtener un resultado similar al siguiente. Observe que su clave y el origen y el destino están integrados en la solicitud de URL. Observe los resultados devueltos por la función como una tupla con los valores de estado, latitud y longitud.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 Graphhopper_parse-json_2.py
Geocoding API URL for Washington, DC:
https://graphhopper.com/api/1/geocode?q=Washington% 2C + DC&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$
• 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
(401, 'null', 'null')
Geocoding API URL for Baltimore, Maryland:
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
(401, 'null', 'null')
○ devasc@labvm:~/labs/devnet-src/graphhopper$
```

Paso 9: Display additional information.

Además del diccionario de puntos con información de latitud y longitud, también puede mostrar otra información relacionada para verificar que las coordenadas sean correctas para la ubicación de entrada.

- a. Muestre los datos JSON nuevamente en JSONview en el navegador Chrome. Observe la información adicional disponible en el diccionario de **coincidencias**.

```
{
  - hits: [
    - {
      + punto: {...},
      + extensión: [...],
      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. Elimine la siguiente declaración de impresión dentro de la función de codificación geográfica. Se reemplazará más adelante en la función.

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

- c. Dentro de la función de codificación geográfica, agregue el **nombre**, el **estado**, el **país** y el **valor** de las nuevas variables. Debido a que las claves de **estado** y **país** no están disponibles

para todas las ubicaciones, las instrucciones **if** se utilizan para asignar las cadenas deseadas para la variable **new_loc** y devolver el valor asignado a **new_loc**. Por ejemplo, el **estado** del valor de clave no se encuentra en el diccionario de **coincidencias** devuelto para **Pekín, 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. Guarda y ejecuta el script **graphhopper_parse-json_2.py** y comprueba que funciona.

```

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')
● devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_2.py
(401, 'null', 'null', 'Washington, D.C.')
(401, 'null', 'null', 'Washington, D.C.')

```

Paso 10: Añadir entrada de usuario para la ubicación inicial y el destino.

Ha utilizado valores estáticos para las variables de ubicación. However, the application requires that the user input these locations. Complete los siguientes pasos para actualizar su aplicación:

- Guarde el script como **graphhopper_parse-json_3.py**.
- Elimine las variables **loc1** y **loc2** actuales.
- Vuelva a escribir el **loc1** y el **loc2** para que estén dentro de un bucle **while** en el que soliciten al usuario la entrada de la ubicación inicial y el destino.. El bucle while permite al usuario continuar realizando solicitudes para diferentes direcciones. Agregue el siguiente código después de la función de codificación geográfica. Asegúrese de que todo el código restante

esté sangrado correctamente dentro del bucle while.

```
while True:
```

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

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

Paso 11: Probar la funcionalidad de entrada del usuario.

- a. Guarda y ejecuta el script **graphhopper_parse-json_3.py** y comprueba que funciona. Solucione el problema de su código, si es necesario Debería obtener una salida similar a la que se muestra a continuación. Para finalizar el programa, escriba **Ctrl+C**. Obtendrá un error de **KeyboardInterrupt** como se muestra en la salida a continuación. Para detener la aplicación con más gracia en ella, agregará la funcionalidad de salir en el siguiente paso.

```
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$
```

Paso 12: Agregue la funcionalidad para salir a la aplicación.

En lugar de forzar el cierre de la aplicación con una interrupción del teclado, añadirá la posibilidad de que el usuario introduzca **q** o **quit** como palabras clave para salir de la aplicación. Complete los siguientes pasos para actualizar su aplicación:

- a. Agregue una instrucción **if** después de cada variable de ubicación para comprobar si el usuario introduce **q** o **quit**, como se muestra a continuación.

```
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)
```

Paso 13: Pruebe la funcionalidad de salir.

Ejecute el script `graphhopper_parse-json_3.py` cuatro veces para probar cada variable de ubicación. Compruebe que tanto `quit` como `q` finalizarán la solicitud. Solucione el problema de su código, si fuera necesario. Usted debería de obtener un resultado similar al siguiente.

```
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$
```

Paso 14: Manejo de errores dentro de la función de codificación geográfica.

Pero, ¿qué sucede si el usuario no ingresó ninguna información de ubicación? Por ejemplo, si el usuario presionó Intro por error y no ingresó ninguna ubicación. ¿Qué sucede si la clave de API estática no es correcta o ya no es válida? ¿Qué sucede si el usuario ingresó una ubicación no válida, como `#`? Todo esto se puede abordar dentro de la función de geocodificación.

- Guarde el script como `graphhopper_parse-json_4.py`.
- Una declaración `while` no puede abordar información de ubicación. Copie y agregue la siguiente declaración resaltada a la función. La función seguirá solicitando la ubicación hasta que el usuario proporcione la información.

```
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})
```

- Guarde y ejecute `graphhopper_parse-json_4.py`. Presione Intro sin ingresar ninguna información de ubicación de inicio para verificar que el bucle `while` funcione correctamente.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_4.py
Starting Location:
Enter the location again: Washington, D.C.
```

Laboratorio - Integrar una API REST en una aplicación Python

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. Para probar la clave de API no válida, copie y pegue una de las URL de la API de codificación geográfica en el campo de dirección del navegador Chromium y agregue un carácter adicional a la clave de API al final de la URL. Recibirá un mensaje similar al siguiente ejemplo:

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

- e. Para mostrar el mensaje de error en el guion, agregue la declaración de impresión resaltada en la función Geocodificación.

```
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. Edite la clave de API agregando un carácter adicional. Guarde y ejecute el guion para verificar el código de estado de respuesta JSON y el mensaje de error se muestra cuando hay una clave no válida.

```
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. Corrija la clave de API y guarde el guion nuevamente antes de pasar a la siguiente parte.
- h. ¿Qué sucede si el usuario ingresó caracteres no válidos, como # o !? ¿U otras entradas no válidas?

```
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. Copie y pegue cualquier URL de la API de geocodificación correcta de la salida anterior en el campo de dirección del navegador Chromium. Por ejemplo, reemplace **Washington% 2C + DC** por **slkdjf**.

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

- j. En este ejemplo, recibió una lista de **resultados** vacía para la solicitud de API correcta.

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

- k. Actualice la condición de prueba de la declaración **if** para que pruebe una solicitud de API correcta y los resultados de los **aciertos** no estén vacíos.

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

- l. Guarde y pruebe **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$
```

Con el operador **y**, tanto **json_status == 200** como **len (json_data ["hits"])! = 0** deben ser verdaderas para que la declaración sea verdadera. En este ejemplo, **len (json_data ["hits"])** es 0, pero **json_status** es 200, por lo que la declaración es falsa. Sin embargo, no se devolvió ningún mensaje de error de la solicitud de la API y **json_data ["mensaje"]** no es válido. Para corregir esto, agregue una instrucción **if** anidada dentro de la parte **else** de la instrucción **if**. Sangra la declaración de impresión para el mensaje de error.

```
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
```

Parte 5: Crear la aplicación de routing de Graphhopper

En la parte anterior, creó una aplicación que acepta la entrada del usuario y devuelve la información de latitud y longitud para una ubicación de origen y destino.

En esta parte, finalizará la aplicación para proporcionar información de ruta entre las ubicaciones de inicio y finalización. Además, también proporcionará información sobre la distancia total recorrida y la duración del viaje. Además, agregará la función del vehículo a su aplicación, donde el usuario puede elegir el método de viaje en tierra.

Paso 1: Cree la URL para la solicitud a la API de routing de Graphhopper.

En este paso, creará la URL para la solicitud de la API de routing si las llamadas a la API de codificación geográfica se realizaron correctamente.

- Guarde el script como **graphhopper_parse-json_5.py**.
- Las declaraciones de **impresión (orig)** e **impresión (de destino)** se pueden eliminar en este momento porque ha verificado los valores de estado, latitud y longitud devueltos por la función de codificación geográfica varias veces.
- La función de codificación geográfica devuelve una tupla con estos tres valores: **json_status**, **latitudesy longitudes**. Agregue una declaración de impresora con un divisor y una declaración **if** para verificar si la función de codificación geográfica devuelve un estado correcto para ambas ubicaciones.

```
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:
```

- Al final del guion, agregue las siguientes líneas de código para solicitar la información de routing para la API de routing de Graphhopper e imprima la URL para su validación mediante JSONView.

- op**: la cadena que proporciona la latitud y la longitud del origen
- dp**: la cadena que proporciona la latitud y la longitud del destino
- paths_url**: URL para solicitar la ruta entre el origen y el destino
- paths_status**: el código de estado devuelto por la URL para solicitar la ruta entre el origen y el destino
- paths_data**: los datos JSON devueltos desde la URL para solicitar la ruta entre el origen y el destino

```
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)
```

- Ejecute su guion **graphhopper_parse-json_5.py**. Compruebe que tanto **quit** como **q** finalizarán

Laboratorio - Integrar una API REST en una aplicación Python

la solicitud. Solucione el problema de su código, si fuera necesario Usted debería de obtener un resultado similar al siguiente. El texto resaltado a continuación es la URL para solicitar la información de routing de la API de Routing.

```
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$
```

Paso 2: Mostrar la información resumida del viaje para incluir la duración y la distancia.

- Copie y pegue la URL de routing en el campo de dirección del navegador Chromium del paso anterior.
- Los resultados solo tienen tres diccionarios raíz: **sugerencias**, **información** y **rutas**. Expanda las **rutas** si es necesario e investigue los datos enriquecidos. La información resaltada a continuación proporciona la duración total de la distancia recorrida.

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

<output omitted>

- La siguiente declaración **if** prueba el estado de respuesta de las API de routing. Cuando el estado de respuesta devuelve 200 para la API de Routing, la variable **paths_data** almacena los datos JSON devueltos por la API de Routing. Se agregan varias declaraciones de **print** que muestran las ubicaciones desde y hasta, así como las teclas de **time** y **distance** para el viaje como cadenas.

Las instrucciones adicionales también incluyen instrucciones de impresión que mostrarán una línea doble antes de la siguiente solicitud de una ubicación inicial. 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. Guarde y ejecute **graphhopper_parse-json_5.py** para ver el siguiente resultado.

```

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. De manera predeterminada, Graphhopper usa el sistema métrico y muestra la distancia en metros. No hay ningún parámetro de solicitud para cambiar los datos al sistema imperial. Por lo tanto, puede convertir su aplicación para mostrar valores imperiales y convertir. Cree dos variables, **miles** y **km**, para convertir los resultados de la unidad de distancia.

```

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

```

- f. Reemplace la declaración de impresión Distancia recorrida como se muestra a continuación que muestra la distancia en ambos sistemas de medición con las variables de **miles** y **km**. 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: {:.1f} miles / {:.1f} km".format(miles, km))
            print("Trip Duration: " + str(paths_data["paths"][0]["time"]) + "
millisec")
        print("=====")

```

- g. Graphhopper informa el tiempo transcurrido en milisegundos. Agregue estas líneas de código para calcular la duración del viaje antes del bloque de declaraciones de impresión para convertir el tiempo

transcurrido de modo que el tiempo de duración del viaje se pueda mostrar en el formato **hh:mm:ss**. El operador de módulo (%) (mod) devuelve el recordatorio, en lugar del cociente.

```
si 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. Reemplace la declaración de impresión de Duración del viaje como se muestra a continuación. Para mostrar la hora en forma de **hh:mm:ss**, los números mostrados deben tener la forma de dos números enteros digitales. Utilice el argumento de formato "{:02d}". Para formatear los valores flotantes a un número entero digital, como se muestra a continuación.

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

Paso 3: Pruebe la funcionalidad de análisis y formato.

Guarda y ejecuta el script **graphhopper_parse-json_5.py** para comprobar que funciona. Solucione el problema de su código, si es necesario Asegúrese de tener todos los paréntesis de apertura y cierre adecuados. Usted debería de obtener un resultado similar al siguiente.

```
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$
```

Paso 4: Inspeccione la lista de instrucciones en los datos de JSON.

- a. Ahora está listo para mostrar las indicaciones paso a paso desde la ubicación inicial hasta el destino. Vuelva al navegador de Chromium donde vio anteriormente la salida en 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. Dentro de la lista de **paths**. The **paths** list includes one big dictionary with most of the JSON data. Busca la lista de **instructions** y contrae cada uno de los siete diccionarios que contiene, como se muestra a

continuación (haz clic en el signo menos «-» para cambiarlo a un signo más «+»). Si está utilizando diferentes ubicaciones, probablemente tendrá una lista de cantidad diferente de instrucciones.

La información resaltada se utilizará como salida para la solicitud completa.

```
{
  + 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. Amplíe el primer diccionario de la lista de **instructions**. Cada diccionario contiene una clave de **text** con un valor, como « Continue onto...», como se muestra a continuación. Debes analizar los datos de JSON para extraer el valor de la clave de **text** que se mostrará dentro de tu aplicación.

```
- 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>
```

Paso 5: Agregue un bucle for para recorrer en iteración los datos JSON de las instrucciones.

Complete los pasos siguientes para actualizar la aplicación y mostrar el valor de la clave de **text**. Para ello, creará un bucle para recorrer en iteración la lista de instrucciones y mostrar el valor del texto de la lista de **instructions** desde la ubicación de inicio hasta el destino.

- a. Guarde el script como **graphhopper_parse-json_6.py**.
- b. Agregue un bucle **for**, resaltado a continuación, después de la segunda sentencia de impresión de doble línea. El bucle **for** recorre en iteración la lista de **instructions** y realiza lo siguiente:
 - 1) Crea dos variables para almacenar el texto y los datos de distancia de los datos JSON de la API de routing
 - 2) Imprime el valor del **text**.
 - 3) Convierte metros en millas.
 - 4) Formats the mile and kilometer value to print only two decimal places with the **"{:2f}"** format function.
- c. Agregue una declaración **print** que muestre una línea doble antes de que la aplicación solicite otra ubicación de inicio, como se muestra a continuación..

Nota: La sentencia de impresión de doble línea no está indentada dentro del bucle for

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

Paso 6: Probar la iteración de JSON.

Guarda y ejecuta el script **graphhopper_parse-json_6.py** para comprobar que funciona. Solucione el problema de su código, si es necesario Usted debería de tener un resultado similar al siguiente

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_6.py
```

Laboratorio - Integrar una API REST en una aplicación Python

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$
```

Paso 7: Informar errores de la API de routing

Ahora está listo para agregar una función para que los usuarios sepan que Graphhopper no puede proporcionar una ruta entre las dos ubicaciones.

Por ejemplo, si un usuario puede ingresar ubicaciones que devolverán una ruta no válida. Si es así, la aplicación muestra la URL y solicita una nueva ubicación de inicio. El usuario no tiene idea de lo que pasó.

- Para provocar un error en la solicitud de la API de enrutamiento sin notificación al usuario, prueba los siguientes valores en tu aplicación. Deberías ver resultados similares.

```
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
```

- Tenga en cuenta que el estado de la API de routing se devolvió como 400. Copia la URL en una pestaña del navegador Chromium. Observe que hay dos entradas de clave / valor en el diccionario. Al agregar una instrucción `else` a la instrucción `if paths_status == 200:`, la aplicación notifica a los usuarios con el mensaje de error recibido de la solicitud fallida de la API de routing de Graphhopper.

```
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("*****")
```

Paso 8: Agregue más modos de transporte.

Con acceso gratuito a Graphhopper, tiene algunos modos de transporte diferentes para elegir. El método predeterminado es el automóvil.

- Guarde el script como **graphhopper_parse-json_7.py**.
- Al agregar las siguientes líneas de código después de **while True :**, permite que los usuarios elijan su modo de transporte terrestre disponible en Graphhopper.

Cuando los usuarios ingresan un método enumerado, el método se asigna a la variable **vehicle**. Si no se encuentra ninguna coincidencia, el **car** se asigna al **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. Busque la variable `paths_url`. Actualice la consulta agregando un nuevo `vehicle` de campo.

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

- d. Busque la declaración impresa con respecto a las instrucciones. Actualice la declaración de impresión con el texto resaltado.

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

Parte 6: Pruebe la funcionalidad completa de la aplicación

Ejecute el script `graphhopper_parse-json_7.py` y compruebe que funciona. Solucione el problema de su código, si es necesario Pruebe todas las características de la aplicación. Usted debería de obtener un resultado similar al siguiente.

```
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$
```