Lab - Parsing JSON with a Python Application (Instructor Version)

1. Objectives

* Obtain a Google API Key.
* Import necessary modules.
* Create API request variables and construct a URL.
* Extract formatted address values.
* Add user input functionality and test for invalid entries.
* Add a quit feature so that the user can end the program.
* Iterate through the JSON data to extract and output long name values.

1. Background / Scenario

In this lab, you will create an application that retrieves JSON data from the Google map geocoding API, parses the data, and formats it for output to the user.

1. Required Resources

* Computer with Python installed according to the **Lab - PC Setup for Workshop**.
* Access to the Internet.
  + 1. Demonstrate the Google map geocoding API application.

Your instructor may demonstrate the program 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.

* + 1. Authenticating a RESTful request.

Before building the application, you will need to obtain a key from Google's developer site, which you will do in the next step. Previously, you used no authentication to access the Google map geocoding API. However, if you make too many calls to that API within a short amount of time, you will get an OVER\_QUERY\_LIMIT response. To prevent this, you authenticate with Google to allow a more generous number of requests.

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

* **None**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  The API resource is public and anybody can place the request. This is the method you have used up to this point.
* **Basic HTTP**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  The username and password are passed to the server in an encoded string. This method is less common than token and OAuth authentication.
* **Token**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  A secret key generally retrieved from the Web API developer portal.
* **Open Authorization (OAuth)**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  An open standard for retrieving an access token from an Identity Provider. The token is then passed with each API call.

For the Google map geocoding API application, you will use token authentication.

* + 1. Get a Google API key.

If you do not have a Google maps geocoding API key, you will need a Google account and then complete the following steps:

* + - 1. Go to: <https://console.developers.google.com>.
      2. Login to your Google account, if necessary, and click **Dashboard > Enable APIs and Services**
      3. For Maps APIs, click **View All**.
      4. Find and enable the **Google Maps Geocoding API**.
      5. Click **Credentials** and create a project with a name of your choice.
      6. Click the **Create credentials** drop down list and choose **API key**.
      7. Copy the API key to a text file and save it for easy reference.

**Note**: Google regularly changes the way its tools work. If the steps above are no longer valid, search for “steps to generate google api key”.

* + 1. Importing Modules

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.

* + - 1. Open a blank script file and save it **08\_parse-json1.py**.
      2. Import the **urllib.parse** and **requests** modules.

import urllib.parse

import requests

* + 1. Create Variables for API Request

The first step in creating your API request is to construct the URL that your program 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
  + **address** - the parameter that you want to access
  + **key** - the Google key you retrieve from the developer website.
    - 1. Create variables to build the URL that will be sent in the request. Copy your Google API key to the key variable.

main\_api = "https://maps.googleapis.com/maps/api/geocode/json?"

address = "san jose"

key = "your\_api\_key"

* + - 1. Combine the three variables **main\_api**, **address**, 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 “+” or “%20”).

url = main\_api + urllib.parse.urlencode({"address": address, "key": key})

* + - 1. 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 **print** statement is used to check the returned data.

json\_data = requests.get(url).json()

print(json\_data)

* + 1. Test the URL request.
       1. Run your **08\_json-parse1.py** script and verify it works. Troubleshoot your code, if necessary. Although your output might be slightly different, you should get a JSON response similar to the following.

{'status': 'OK', 'results': [{'types': ['locality', 'political'], 'formatted\_address': 'San Jose, CA, USA', 'geometry': {'bounds': {'northeast': {'lat': 37.4695381, 'lng': -121.589154}, 'southwest': {'lat': 37.124493, 'lng': -122.0456719}}, 'viewport': {'northeast': {'lat': 37.4695381, 'lng': -121.589154}, 'southwest': {'lat': 37.124493, 'lng': -122.0456719}}, 'location\_type': 'APPROXIMATE', 'location': {'lat': 37.3382082, 'lng': -121.8863286}}, 'address\_components': [{'long\_name': 'San Jose', 'types': ['locality', 'political'], 'short\_name': 'San Jose'}, {'long\_name': 'Santa Clara County', 'types': ['administrative\_area\_level\_2', 'political'], 'short\_name': 'Santa Clara County'}, {'long\_name': 'California', 'types': ['administrative\_area\_level\_1', 'political'], 'short\_name': 'CA'}, {'long\_name': 'United States', 'types': ['country', 'political'], 'short\_name': 'US'}], 'place\_id': 'ChIJ9T\_5iuTKj4ARe3GfygqMnbk'}]}

>>>

* + - 1. Change the address variable and rerun the script to get different results.

**Note**: If you get a 'REQUEST\_DENIED' message, return to [Google developer's console](https://console.developers.google.com/) and make sure your API key is enabled with no restrictions and that you correctly copied the key. You can also request a new, fresh API key.

* + 1. 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.

* + - 1. Save your script as **08\_json-parse2.py**.
      2. Delete the **print(json\_data)** statement as you no longer need to test that the request is properly formatted.
      3. 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 program.
* Print the status of the request, which would normally be **OK** if the request is successful.

print(url)

json\_status = json\_data["status"]

print("API Status: " + json\_status)

* + 1. Test status and URL print commands.

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

https://maps.googleapis.com/maps/api/geocode/json?address=san+jose&key='your\_api\_key'

API Status: OK

>>>

* + 1. Extract the formatted address value.

Now you are ready to parse the JSON data and extract content you are interested in displaying to the user. The first piece of data you want to extract is the value for the **formatted\_address**. The **formatted\_address** value is embedded three levels down in the JSON data, as shown below.



To access the data returned to the **json\_data** variable, you first reference the dictionary entry for the results key. The value of the **results** key is a list. You are interested in the first entry in the list, which you access by referencing its index, **[0]**. The first entry in the results list is another dictionary, so your access the formatted address value by referencing its key name, **formatted\_address**.

Complete the following steps to update your application:

* + - 1. Save your script as **08\_json-parse3.py**.
      2. Create a variable to extract the formatted address.
      3. Print the **formatted\_address** value.

formatted\_address = json\_data["results"][0]["formatted\_address"]

print(formatted\_address)

* + 1. Test the formatted address variable.

Run your **08\_json-parse3.py** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to the following.

https://maps.googleapis.com/maps/api/geocode/json?address=sjc&key='your\_api\_key'

API Status: OK

Norman Y. Mineta San Jose International Airport (SJC), 1701 Airport Blvd, San Jose, CA 95110, USA

>>>

* + 1. Add User Input for Address

Up to this point, you have used San Jose as the static value for the address variable. However, the application requires that the user input the address value. Complete the following steps to update your application:

* + - 1. Save your script as **08\_json-parse4.py**.
      2. Delete the current address variable.
      3. Rewrite the **address** variable with a while loop before the **url** variable to request user input for address. The while loop will allow the user to continue to make requests to different addresses.
      4. Be sure all the remaining code is indented within the while loop.

while True:

address = input("Address: ")

url = main\_api + urllib.parse.urlencode({"address": address, "key": key})

print(url)

json\_data = requests.get(url).json()

json\_status = json\_data["status"]

print("API Status: " + json\_status)

formatted\_address = json\_data["results"][0]["formatted\_address"]

print(formatted\_address)

* + 1. Test user input functionality.

Run your **08\_json-parse4.py** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to what is shown below. You will add quit functionality later. For now, enter **Ctrl+C** to quit the program.

Address: **sjc**

https://maps.googleapis.com/maps/api/geocode/json?address=sjc&key='your\_api\_key'

API Status: OK

Norman Y. Mineta San Jose International Airport (SJC), 1701 Airport Blvd, San Jose, CA 95110, USA

Address: **lax**

https://maps.googleapis.com/maps/api/geocode/json?address=lax&key='your\_api\_key'

API Status: OK

Los Angeles International Airport (LAX), 1 World Way, Los Angeles, CA 90045, USA

Address:

* + 1. Check for invalid user input.

Now that the user can enter an address value, you need to check for invalid entries. Currently, the application will return a status of ZERO\_RESULTS. However, the application will try to set the formatted\_address variable and fail, as shown below.

Address: **;lksdff**

https://maps.googleapis.com/maps/api/geocode/json?address=%3Blksdff&key='your\_api\_key'

API Status: ZERO\_RESULTS

Traceback (most recent call last):

File "/home/user/08\_parse-json5.py", line 25, in <module>

formatted\_address = json\_data["results"][0]["formatted\_address"]

IndexError: list index out of range

**Note**: The output has been modified to remove the API key.

To check for invalid user input, complete the following steps to update your application:

* + - 1. Save your script as **08\_json-parse5.py**.
      2. Add an if statement in the while loop to check for an invalid entry immediately after you print the **json\_status**, as shown below.

while True:

address = input("Address: ")

url = main\_api + urllib.parse.urlencode({"address": address, "key": key})

print(url)

json\_data = requests.get(url).json()

json\_status = json\_data["status"]

print("API Status: " + json\_status)

if json\_status == "OK":

formatted\_address = json\_data["results"][0]["formatted\_address"]

print(formatted\_address)

* + 1. Test for invalid user input.

Run your **08\_json-parse5.py** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to what is shown below. Enter **Ctrl+C** to quit the program.

Address: **k;lksdjf**

https://maps.googleapis.com/maps/api/geocode/json?address=k%3Blksdjf&key='your\_api\_key'

API Status: ZERO\_RESULTS

Address:

* + 1. Add the quit functionality to the application

Instead of entering Ctrl+C to quit the program, you will add the ability for the user to enter **q** or **quit** as keywords to quit the program. Complete the following steps to update your application:

* + - 1. Save your script as **08\_json-parse6.py**.
      2. Add an if statement after the address variable to check if the user enters **q** or **quit**, as shown below.

while True:

address = input("Address: ")

if address == "quit" or address == "q":

break

* + 1. Test the quit functionality.

Run your **08\_json-parse6.py** script twice to verify that both **quit** and **q** will end the application. Troubleshoot your code, if necessary. You should get output similar to the following.

Address: **quit**

>>>

Address: **q**

>>>

* + 1. Add a For Loop to Iterate Through the JSON Data

Recall that the final application prints the value for the **long\_name** keys from the dictionaries in the **address\_components** list. As a reminder, the figure below shows the JSON displayed in a web browser.



The application prints each of the four long\_name values on a separate line, as highlighted below.

========= RESTART: /home/user/ /08\_parse-json\_sol.py =========

Address: **San Jose**

https://maps.googleapis.com/maps/api/geocode/json?address=San+Jose&key='your\_api\_key'

API Status: OK

San Jose

Santa Clara County

California

United States

San Jose, CA, USA

Address:

The **address\_components** list can have any number of **long\_name** values. For example, the address San Jose has four **long\_name** values. But, the airport code for San Jose International Airport (SJC) has eight **long\_name** values.

To display all the **long\_name** values, you need to iterate through the **address\_component** list, extracting each **long\_name** value until there are no more left.

Complete the following steps to update your application:

* + - 1. Save your script as **08\_json-parse7.py**.
      2. Add a for loop within the if statement that checks for the JSON status.
      3. Add a print statement to print each long\_name value found
      4. To better distinguish parts of the output, use the **\n** special character to add a blank line to the JSON status print statement and the formatted\_address print statement.

json\_status = json\_data["status"]

print("API Status: " + json\_status + "\n")

if json\_status == "OK":

for each in json\_data["results"][0]["address\_components"]:

print(each["long\_name"])

formatted\_address = json\_data["results"][0]["formatted\_address"]

print("\n" + formatted\_address)

* + 1. Activity - Test the JSON Iteration and full program functionality.

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

Address: **sjc**

https://maps.googleapis.com/maps/api/geocode/json?address=sjc&key='your\_api\_key'

API Status: OK

Norman Y. Mineta San Jose International Airport

1701

Airport Boulevard

San Jose

Santa Clara County

California

United States

95110

Norman Y. Mineta San Jose International Airport (SJC), 1701 Airport Blvd, San Jose, CA 95110, USA

Address: