

# -API Scavenger Hunt-

- Assignment by Shounak Kulkarni (CUID – C56298850) -

<https://github.com/ShounaKulkarni/api-scamenger-hunt>

## Task1: Solution.

### 1) current weather for London, United Kingdom –

#### Code & output screenshot –

```
In [12]: import requests

# enter API
api_key = 'dfcfd3203bac0f1e51334953c50081c7'

# Current weather for London
london_weather_url = f"http://api.openweathermap.org/data/2.5/weather?q=London,uk&exclude=minutely,hourly,daily,alerts&appid={api_key}"
london_weather_response = requests.get(london_weather_url)
london_weather = london_weather_response.json()

# 5-day forecast for Tokyo
tokyo_forecast_url = f"http://api.openweathermap.org/data/2.5/forecast?q=Tokyo,jp&exclude=minutely,hourly,daily,alerts&appid={api_key}"
tokyo_forecast_response = requests.get(tokyo_forecast_url)
tokyo_forecast = tokyo_forecast_response.json()

# Print the results or process them as needed
print(london_weather)
#print(tokyo_forecast)

{'coord': {'lon': -0.1257, 'lat': 51.5085}, 'weather': [{'id': 800, 'main': 'Clear', 'description': 'clear sky', 'icon': '01d'}], 'base': 'stations', 'main': {'temp': 285.57, 'feels_like': 284.8, 'temp_min': 283.96, 'temp_max': 286.63, 'pressure': 982, 'humidity': 74}, 'visibility': 10000, 'wind': {'speed': 3.58, 'deg': 228, 'gust': 6.26}, 'clouds': {'all': 5}, 'dt': 1699196970, 'sys': {'type': 2, 'id': 2075535, 'country': 'GB', 'sunrise': 1699167617, 'sunset': 1699201670}, 'timezone': 0, 'id': 2643743, 'name': 'London', 'cod': 200}
```

## 2) 5-day forecast for Tokyo, Japan –

### Code & output screenshot –

It is not possible to display 5 day data all within a single screenshot. However, I have compiled the entire output into an HTML file titled **task1.html** below. The complete output is also accessible through a Jupyter Notebook file on GitHub. The link to this repository is provided on the first page of this document.



task1.html

```
In [13]: import requests

# enter API
api_key = 'dfcfd3203bac0f1e51334953c50081c7'

# Current weather for London
london_weather_url = f"http://api.openweathermap.org/data/2.5/weather?q=London,uk&exclude=minutely,hourly,daily,alerts&appid={api_key}"
london_weather_response = requests.get(london_weather_url)
london_weather = london_weather_response.json()

# 5-day forecast for Tokyo
tokyo_forecast_url = f"http://api.openweathermap.org/data/2.5/forecast?q=Tokyo,jp&exclude=minutely,hourly,daily,alerts&appid={api_key}"
tokyo_forecast_response = requests.get(tokyo_forecast_url)
tokyo_forecast = tokyo_forecast_response.json()

# Print the results
print(london_weather)
print(tokyo_forecast)
```

```
{'cod': '200', 'message': 0, 'cnt': 40, 'list': [{'dt': '1699207200', 'main': {'temp': 292.09, 'feels_like': 292.26, 'temp_min': 292.89, 'temp_max': 292.94, 'pressure': 1024, 'sea_level': 1024, 'grnd_level': 1019, 'humidity': 85, 'temp_kf': -0.85}, 'weather': [{'id': 500, 'main': 'Rain', 'description': 'light rain', 'icon': '10n'}], 'clouds': {'all': 56}, 'wind': {'speed': 2.33, 'deg': 174, 'gust': 5.29}, 'visibility': 10000, 'pop': 0.39, 'rain': {'3h': 0.15}, 'sys': {'pod': 'n'}, 'dt_txt': '2023-11-05 18:00:00'}, {'dt': '1699218000', 'main': {'temp': 292.63, 'feels_like': 292.75, 'temp_min': 292.63, 'temp_max': 293.11, 'pressure': 1023, 'sea_level': 1023, 'grnd_level': 1018, 'humidity': 81, 'temp_kf': -0.48}, 'weather': [{'id': 500, 'main': 'Rain', 'description': 'light rain', 'icon': '10n'}], 'clouds': {'all': 40}, 'wind': {'speed': 4.51, 'deg': 191, 'gust': 8.64}, 'visibility': 10000, 'pop': 0.3, 'rain': {'3h': 0.19}, 'sys': {'pod': 'n'}, 'dt_txt': '2023-11-05 21:00:00'}, {'dt': '1699228800', 'main': {'temp': 295.74, 'feels_like': 295.7, 'temp_min': 295.74, 'temp_max': 295.74, 'pressure': 1022, 'sea_level': 1022, 'grnd_level': 1017, 'humidity': 63, 'temp_kf': 0}, 'weather': [{'id': 802, 'main': 'Clouds', 'description': 'scattered clouds', 'icon': '03d'}], 'clouds': {'all': 29}, 'wind': {'speed': 7.28, 'deg': 191, 'gust': 10.58}, 'visibility': 10000, 'pop': 0.18, 'sys': {'pod': 'd'}, 'dt_txt': '2023-11-06 00:00:00'}, {'dt': '1699239600', 'main': {'temp': 297.23, 'feels_like': 297.16, 'temp_min': 297.23, 'temp_max': 297.23, 'pressure': 1020, 'sea_level': 1020, 'grnd_level': 1015, 'humidity': 56, 'temp_kf': 0}, 'weather': [{'id': 803, 'main': 'Clouds', 'description': 'broken clouds', 'icon': '04d'}], 'clouds': {'all': 83}, 'wind': {'speed': 9.27, 'deg': 189, 'gust': 12.8}, 'visibility': 10000, 'pop': 0, 'sys': {'pod': 'd'}, 'dt_txt': '2023-11-06 03:00:00'}, {'dt': '1699250400', 'main': {'temp': 296.83, 'feels_like': 296.9, 'temp_min': 296.83, 'temp_max': 296.83, 'pressure': 1018, 'sea_level': 1018, 'grnd_level': 1013, 'humidity': 63, 'temp_kf': 0}, 'weather': [{'id': 804, 'main': 'Clouds', 'description': 'overcast clouds', 'icon': '04d'}], 'clouds': {'all': 91}, 'wind': {'speed': 9.53, 'deg': 189, 'gust': 13.86}, 'visibility': 10000, 'pop': 0.03, 'sys': {'pod':
```

In [ ]: 

## **Reflection on Using the OpenWeatherMap API**

My experience with the OpenWeatherMap API was quite positive, marked by the ease of obtaining the API key and the simplicity of the API's interface. Registering on their platform and navigating through the API documentation was straightforward, allowing me to quickly move on to making actual API calls. The documentation was quite clear and comprehensive, which made it easy for me to compose the request and retrieve the current weather for London. Because of the JSON response's organization, it was simple to extract the pertinent meteorological data.

The procedure was as simple to utilize to retrieve the Tokyo 5-day forecast. With just a GET request, users could receive a comprehensive forecast from the API that includes numerous weather indicators like temperature, humidity, and precipitation. The level of detail in the data for every day was astounding, providing insights that might be used to everything from agricultural platforms that depend on weather forecasts to trip planning apps.

When considering the OpenWeatherMap API's overall capability, a wide range of capabilities that can meet different purposes are provided. With a daily cap of 1,000 free API requests, developers have a great chance to incorporate meteorological data into their apps without worrying about sudden expenses. This API has a wide range of possible uses, from augmenting event planning platforms with weather forecasts to incorporating weather data into smart home systems. The OpenWeatherMap API is an effective tool for developers wishing to incorporate real-time weather elements into their applications because of its simplicity of use and the breadth of data it offers.

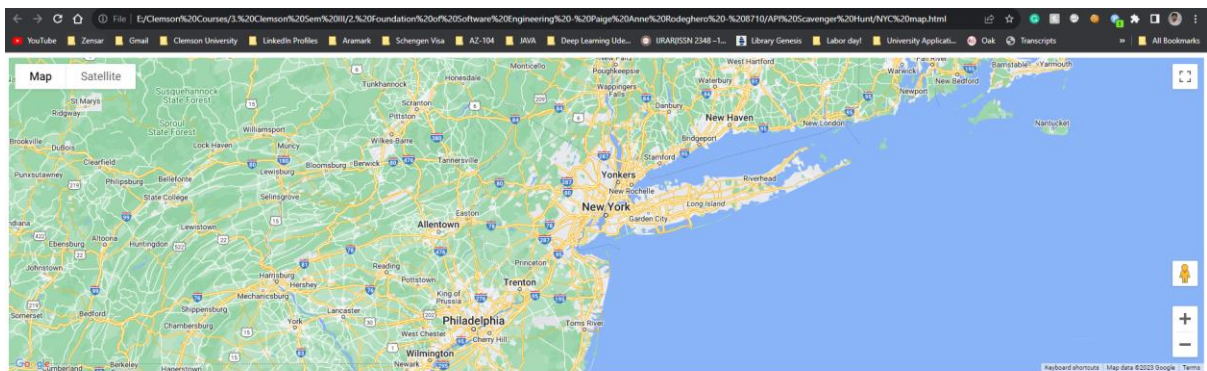
## Task2: Solution –

### 1) Map centered on New York City, USA -

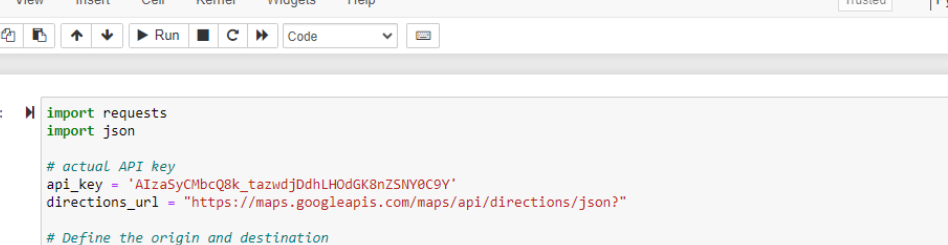
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Simple Map</title>
5 <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCMbcQ8k_tazwdjDdhLH0dGK8nZSNY0C9Y&callback=initMap" async defer></script>
6 </script>
7 var map;
8 function initMap() {
9   map = new google.maps.Map(document.getElementById('map'), {
10     center: {lat: 40.7128, lng: -74.0060},
11     zoom: 8
12   });
13 }
14 </script>
15 </head>
16 <body>
17 <div id="map" style="height: 500px; width: 100%;"></div>
18 </body>
19 </html>
20
```



NYC map.html



2) shortest route by car between San Francisco, USA, and Los Angeles, USA –



The screenshot shows a Jupyter Notebook interface. At the top, the title bar reads "jupyter task2 Last Checkpoint: 4 minutes ago (autosaved)". Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Trusted" and "Python 3 (ipykernel)". Below the menu bar is a toolbar with icons for saving, adding, undo, redo, and running code. The main area of the notebook displays a Python script in a code cell, which is currently selected. The script is as follows:

```
In [1]: import requests
import json

# actual API key
api_key = 'AIzaSyCMbcQ8k_tazwdjDdhLH0dGK8nZSNY0C9Y'
directions_url = "https://maps.googleapis.com/maps/api/directions/json?"

# Define the origin and destination
origin = "San Francisco, USA"
destination = "Los Angeles, USA"

# Define the parameters for the request
params = {
    'origin': origin,
    'destination': destination,
    'mode': 'driving',
    'key': api_key
}

# Make the get request
response = requests.get(directions_url, params=params)

# Load the response into a JSON object
directions = response.json()

# Print the directions or process them as needed
print(json.dumps(directions, indent=2))
```

**OUTPUT** – However, I have compiled the entire output into an HTML file titled **task2.html** below. The complete output is also accessible through a Jupyter Notebook file on GitHub. The link to this repository is provided on the first page of this document.



task2.html

```

    "duration": {
      "text": "3 mins",
      "value": 153
    },
    "end_location": {
      "lat": 37.7692346,
      "lng": -122.4178853
    },
    "html_instructions": "Head <b>south</b> on <b>S Van Ness Ave</b> toward <b>12th St</b>.",
    "polyline": {
      "points": "e|peFt_ejVLCdGj@KdBe@z@n1B_e@I@e@daUPEPGNGLCdBa@T@T@Qd@KNCIXLEDcROJIJIFC@?B?B?B?D@p@DD?@?@?NBf@@?RB@?D?J@DD?n@BZB"
    },
    "start_location": {
      "lat": 37.7749134,
      "lng": -122.4193088
    },
    "travel_mode": "DRIVING"
  },
  {

```

```
    "duration": {
      "text": "1 min",
      "value": 34
    },
    "end_location": {
      "lat": 37.7696292,
      "lng": -122.4170769
    },
    "html_instructions": "Slight <b>right</b> onto the <b>US-101 S</b> ramp to <b>I-80 E</b> <b>Oakland</b> <b>San Jose</b>.",
    "maneuver": "ramp-right",
    "polyline": {
      "points": "uxoeFxdjVLLNJBBDHLDf@DBD@H@D?D@B?F?B@@?DAD?DAH?DAFAFADEFCHCDEDCBEDGDEBC@CBE?C@I?I?G?GAEAECCA CAIGGGEGACAACICGCGAI?AAEAI?I?K?K@MBYDa@Dc@Fw@F}@?C?CIS"
    },
    "start_location": {
      "lat": 37.7692346,
      "lng": -122.4178853
    },
    "travel_mode": "DRIVING"
```

```
    "end_location": {
      "lat": 37.769057,
      "lng": -122.4092529
    },
    "html_instructions": "Continue onto <b>US-101 S</b> <b>Central Fwy</b>.",
    "polyline": {
      "points": "e{oeFvqjVB_@Be@B]K?IS@o@q@q@_@?g@eB@y@q@q@}@@{@@s@?Y?[@m@?GBYB@O?c@@]?[a@Bc@Bq@JyBFq@Fu@F k@BY@KToB"
    },
    "start_location": {
      "lat": 37.7696292,
      "lng": -122.4170769
    },
    "duration": {
      "text": "8 mins",
      "value": 490
    },
    "end_location": {
      "lat": 37.8251891,
      "lng": -122.3047588
    },
    "html_instructions": "Take the exit on the <b>left</b> onto <b>I-80 E</b> toward <b>Bay Brg</b> <b>Oakland</b>.",
    "maneuver": "ramp-left",
```

```
    "text": "1 min",
    "value": 57
  },
  "end_location": {
    "lat": 37.8263412,
    "lng": -122.2890091
  },
  "html_instructions": "Take exit <b>88</b> for <b>I-580 E</b> toward <b>Oakland</b> <b>US-24</b>.",
  "maneuver": "ramp-right",
  "polyline": {
    "points": "mvzeFvsniv@{Gm@[_C?AIy@CSCUE]Ec@E]Ea@CYAGC]E]Ea@E_@CHGg@E]CYGa@UsBGi@Ea@?AE[COCOE[AG?EE[G_@CW CSKq@E]CWCIEc@G[I]@M]@Km@Ic@G]I]GSCGI_@I[KYK_@IYCKM_@I[EMGSEWEUEUCSCOCOSAUCUAOAW?YAOAg@aa@Bu@Dm@Fi@Hg@F]F[FWFUFUPi@HSPe@ HWFOFOBMFODQBKH]H[Le@He@HK@D]BUHs@BaA@e@"
  },
  "duration": {
    "text": "41 mins",
    "value": 2487
  },
  "end_location": {
    "lat": 37.7417513,
    "lng": -121.573913
  },
  "html_instructions": "Continue onto <b>I-580 E</b>.",
  "polyline": {
```

```
    "lng": -122.2890091
  },
  "travel_mode": "DRIVING"
},
{
  "distance": {
    "text": "16.9 mi",
    "value": 27220
  },
  "duration": {
    "text": "15 mins",
    "value": 874
  },
  "end_location": {
    "lat": 37.5909837,
    "lng": -121.3339934
  },
  "html_instructions": "Keep <b>right</b> at the fork to stay on <b>I-580 E</b>, follow signs for <b>Intersta te 580</b> <b>Interstate 5 S</b> <b>Fresno</b> <b>Los Angeles</b>.",
  "maneuver": "fork-right"
```

```
    },
    "distance": {
      "text": "280 mi",
      "value": 450627
    },
    "duration": {
      "text": "4 hours 8 mins",
      "value": 14865
    },
    "end_location": {
      "lat": 34.3653479,
      "lng": -118.5566321
    },
    "html_instructions": "Continue onto <b>I-5 S</b>",
    "polyline": {
```

```
      "lat": 34.3653479,
      "lng": -118.5556726
    },
    "html_instructions": "Take exit <b>166</b> for <b>Calgrove Blvd</b>",
    "maneuver": "ramp-right",
    "polyline": {
      "points": "m~vpE|qrrUf@H@AFCNGZ0?b@Q1Aa@LEB?PE~@UdAMDA\\CB?PALAH?f@AX?F?N?PALALCLEp@UZH"
    },
    "start_location": {
      "lat": 34.3653479,
      "lng": -118.5566321
    },
    "travel_mode": "DRIVING"
```

```
    "lat": 34.3602771,
    "lng": -118.5560666
  },
  "html_instructions": "Turn <b>right</b> onto <b>Calgrove Blvd</b>",
  "maneuver": "turn-right",
  "polyline": {
    "points": "{hvpE|krrU`@X\\RLFJDTHB@VFXF\\DX@XA\\CJA`@G"
  },
  "start_location": {
    "lat": 34.361903,
    "lng": -118.5556726
  },
```

```
    "duration": {
      "text": "1 min",
      "value": 63
    },
    "end_location": {
      "lat": 34.32358840000001,
      "lng": -118.5029981
    },
    "html_instructions": "Continue onto <b>San Fernando Rd</b>",
    "polyline": {
      "points": "krppEvvhrUTKPIRIDAFcb@OTIZKfA]p@Uz@Y\\Kp@U`A[XIpAc@DAnGsBdCy@nBo@tAe@bA]\\IFCLCF?JAP?L@j@B~AJN@JB\\DXD`@EZERCNCBAPERIPKDAJEHGFEFGFIHK"
    },
    "travel_mode": "DRIVING"
  },
  {
    "distance": {
      "text": "0.9 mi",
      "value": 1450
    },
    "duration": {
      "text": "2 mins",
      "value": 92
    },
    "end_location": {
      "lat": 34.3153247,
      "lng": -118.4912417
    },
    "html_instructions": "Continue straight to stay on <b>San Fernando Rd</b>",
    "maneuver": "straight",
    "polyline": {
      "text": "0.9 mi",
      "value": 1426
    },
    "duration": {
      "text": "1 min",
      "value": 71
    },
    "end_location": {
      "lat": 34.078576,
      "lng": -118.228452
    },
    "html_instructions": "Take the <b>CA-110 S</b> exit toward <b>Los Angeles</b>",
    "maneuver": "ramp-right",
    "polyline": {
      "text": "1 min",
      "value": 59
    },
    "end_location": {
      "lat": 34.0695669,
      "lng": -118.2363335
    },
    "html_instructions": "Merge onto <b>CA-110</b>",
    "maneuver": "merge",
    "polyline": {
```

```
    "lat": 34.0667057,  
    "lng": -118.2375889  
  },  
  "html_instructions": "Take exit <b>24C</b> on the <b>left</b> for <b>Hill St</b> toward <b>Civic Ctr</b>",  
  "maneuver": "ramp-left",  
  "polyline": {  
    "points": "ye`nE`tpUTE@?JFXLRJd@NJDNFNFF@XJXJB@`@N@?RH`A\\`@NRH@?VJFBD@XLHDF1@THBJF@?DC"  
  },  
  "duration": {  
    "text": "3 mins",  
    "value": 178  
  },  
  "end_location": {  
    "lat": 34.0564992,  
    "lng": -118.2445046  
  },  
  "html_instructions": "Continue onto <b>N Hill St</b>",  
  "text": "1 min",  
  "value": 77  
},  
"end_location": {  
  "lat": 34.0549067,  
  "lng": -118.2426508  
},  
"html_instructions": "Turn <b>left</b> onto <b>W Temple St</b>",  
"maneuver": "turn-left",  
"polyline": {  
  "points": "ctznEbsupUNQz@gA@AJM-@kAV[NQb@i@r@{@PQPU"
```

## Reflection on Using the Google Maps API

Diving into the Google Maps API was an engaging and educational journey. Thanks to Google's comprehensive developer documentation, getting an API key was a simple task. It only took a few lines of code to implement a map centered on New York City, demonstrating how user-friendly the API is. It was simple to adapt the look and behavior of the map to the demands of the current assignment thanks to Google's extensive documentation. The development process was made easier by the Google Cloud Platform's user-friendly interface and the responsiveness of the API.

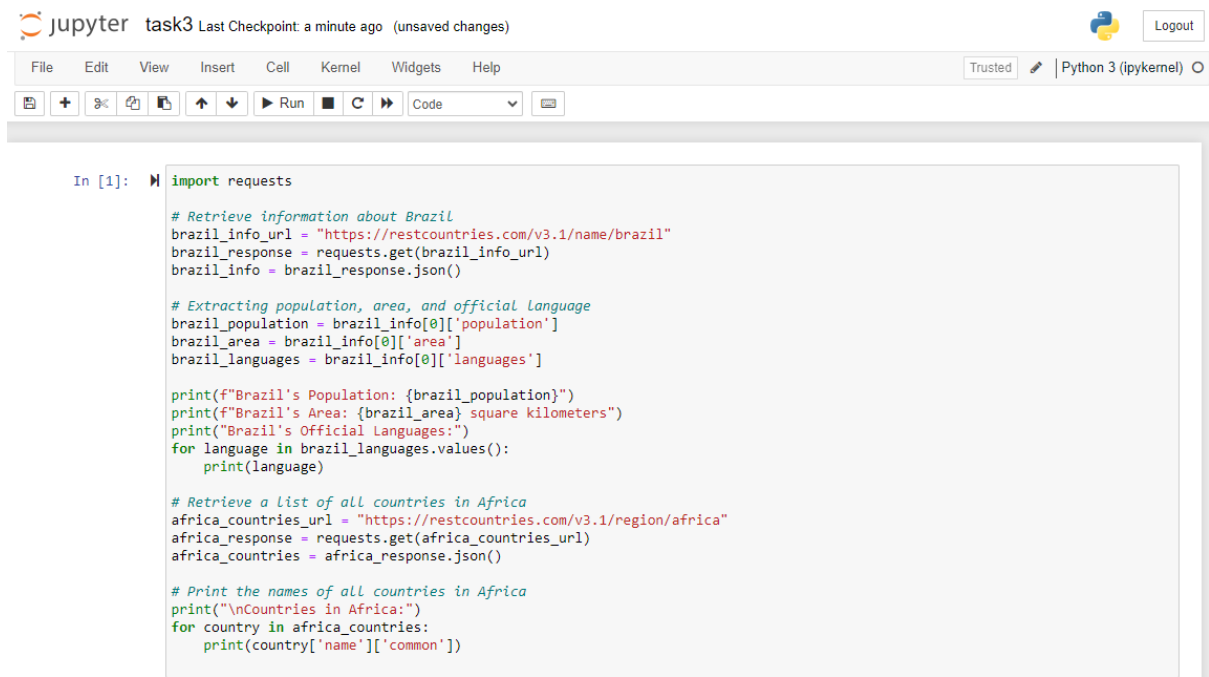
The challenge of determining the quickest road route between San Francisco and Los Angeles by car demonstrated the API's strong route optimization capabilities. Building navigational aids or logistics software may benefit greatly from the precise instructions and real-time traffic data provided by the Directions API, which is a component of Google Maps services. User experiences in delivery services, trip planning, and other areas can be revolutionized by the ability to incorporate such intricate mapping and routing features into applications.

Reflecting on the overall experience, Google Maps API stands out for its extensive features and reliability. The potential applications are vast, ranging from real estate platforms showcasing property locations to social apps suggesting meeting points based on user location. The ease of use, combined with the powerful capabilities of the API, makes it an indispensable tool for developers looking to incorporate geographical mapping and intelligent routing into their applications. The experience has left me with a deeper appreciation for the sophistication of Google's mapping services and the myriad ways they can be leveraged in software development.



## Task3: Solution –

### Code snippet –



The image shows a Jupyter Notebook interface. At the top, it says "jupyter task3 Last Checkpoint: a minute ago (unsaved changes)". The top bar includes a "Logout" button and a "Python 3 (ipykernel)" label. The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding cells, running, and other functions. The main area is a code cell labeled "In [1]:". It contains the following Python code:

```
import requests

# Retrieve information about Brazil
brazil_info_url = "https://restcountries.com/v3.1/name/brazil"
brazil_response = requests.get(brazil_info_url)
brazil_info = brazil_response.json()

# Extracting population, area, and official language
brazil_population = brazil_info[0]['population']
brazil_area = brazil_info[0]['area']
brazil_languages = brazil_info[0]['languages']

print(f"Brazil's Population: {brazil_population}")
print(f"Brazil's Area: {brazil_area} square kilometers")
print("Brazil's Official Languages:")
for language in brazil_languages.values():
    print(language)

# Retrieve a list of all countries in Africa
africa_countries_url = "https://restcountries.com/v3.1/region/africa"
africa_response = requests.get(africa_countries_url)
africa_countries = africa_response.json()


# Print the names of all countries in Africa
print("\nCountries in Africa:")
for country in africa_countries:
    print(country['name']['common'])
```

## Output Screenshot –

- 1) information about Brazil, including its population, area, and official language -

```
Brazil's Population: 212559409
Brazil's Area: 8515767.0 square kilometers
Brazil's Official Languages:
Portuguese
```

## 2) list of all countries in Africa-

jupyter task3 Last Checkpoint: 2 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

Run Code

```
Countries in Africa:
Malawi
Cameroon
Nigeria
Western Sahara
Lesotho
Mayotte
Rwanda
Sierra Leone
Benin
Ghana
Central African Republic
Kenya
Egypt
Tunisia
Sudan
Zimbabwe
Togo
British Indian Ocean Territory
Tanzania
Gabon
Burundi
Ethiopia
Madagascar
Republic of the Congo
Gambia
Guinea
Zambia
Eritrea
Burkina Faso
Ivory Coast
Cape Verde
Guinea-Bissau
Mali
South Sudan
Seychelles
Chad
Djibouti
Namibia
Mozambique
Mauritius
Saint Helena, Ascension and Tristan da Cunha
Comoros
Equatorial Guinea
Uganda
Botswana
Libya
Algeria
São Tomé and Príncipe
Angola
Niger
Réunion
Senegal
Morocco
Somalia
DR Congo
Mauritania
South Africa
Liberia
Eswatini
```

In [ ]: ▶ |

## Reflection on Using the REST Countries API -


My experience using the REST Countries API was quite simple. The absence of an API key requirement facilitated immediate access, allowing me to focus on exploring the API's capabilities without preliminary hurdles. The documentation was very easy to use and included clear instructions on how to utilize the endpoints, which made getting data about the languages and demographics of Brazil a simple process. The developer-friendly design of the API was demonstrated by how simple it was to extract and parse the data from its neatly organized JSON response.

The API's quick delivery of a list of African nations demonstrated its potential for use in applications like market analysis tools or educational platforms that need to segment data based on area. The vast dataset made available by the simple API requests might be used for a variety of purposes, such as demographic research, content localization, or even as the basis for an educational tool that focuses on geography.

In conclusion, I was rather satisfied with the REST Countries API's usability and range of data it offers. Because of its simple methodology and extensive data set, developers have a plethora of options when it comes to incorporating a global perspective into their apps. My knowledge of RESTful APIs has expanded because of the experience, and it has also given me ideas for new applications across a range of industries.

## Task4: Solution –

## Source code –

```
jupyter task4 Last Checkpoint: 2 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [1]: import requests


# actual API key
api_key = 'pr_1fa20cf4b1e944c0b31ba3b9e1f79497'

# Function to convert currencies
def convert_currency(amount, from_currency, to_currency, api_key):
    url = f"https://free.currconv.com/api/v7/convert?q={from_currency}_{to_currency}&compact=ultra&apiKey={api_key}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        rate = data[f"{from_currency}_{to_currency}"]
        return rate * amount
    else:
        print("Error:", response.status_code, response.text)
        return None

# Convert 100 USD to EUR
usd_to_eur = convert_currency(100, 'USD', 'EUR', api_key)
print(f"100 USD is {usd_to_eur} EUR")

# Convert 1000 JPY to GBP
jpy_to_gbp = convert_currency(1000, 'JPY', 'GBP', api_key)
print(f"1000 JPY is {jpy_to_gbp} GBP")
```

## OUTPUT –

```
jupyter task4 Last Checkpoint: 2 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [1]: import requests

# actual API key
api_key = 'pr_1fa20cf4b1e944c0b31ba3b9e1f79497'

# Function to convert currencies
def convert_currency(amount, from_currency, to_currency, api_key):
    url = f"https://free.currconv.com/api/v7/convert?q={from_currency}_{to_currency}&compact=ultra&apiKey={api_key}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        rate = data[f"{from_currency}_{to_currency}"]
        return rate * amount
    else:
        print("Error:", response.status_code, response.text)
        return None

# Convert 100 USD to EUR
usd_to_eur = convert_currency(100, 'USD', 'EUR', api_key)
print(f"100 USD is {usd_to_eur} EUR")

# Convert 1000 JPY to GBP
jpy_to_gbp = convert_currency(1000, 'JPY', 'GBP', api_key)
print(f"1000 JPY is {jpy_to_gbp} GBP")

100 USD is 93.312 EUR
1000 JPY is 5.397 GBP
```

#### **\*\*\*\*\*Reflection of task 4:**

After completing the tasks,

The Currency Converter API was extremely easy to use, with clear, well-documented endpoints that made integrating it with my Python script a snap. From receiving the API key to carrying out currency conversions, the procedure was easy to follow and included clear examples to ensure a hassle-free experience. Fast API response times and simply comprehensible data formats are essential for developers who want to provide functionality with the least amount of complexity.

Application-wise, the API provides fundamental features that are readily extensible to real-world use cases, such real-time currency conversion for financial services or e-commerce platforms. Even with its restrictions, the free tier is a great place to start learning about how financial data is integrated and used in applications, and its premium plans offer the possibility to expand into even more sophisticated features. I could not find any free API key so I did purchase a 6\$ subscription as deadline was approaching and free API key needed 3-5 business day.