

-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': 292.09, '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. Retrieving the current weather for London required minimal effort, thanks to the clear and concise documentation that guided me through constructing the request. The JSON response was well-organized, enabling easy extraction of the relevant weather data.

When it came to fetching the 5-day forecast for Tokyo, the process was just as user-friendly. The API provided a detailed forecast that included a variety of weather indicators such as temperature, humidity, and precipitation, all of which could be accessed with a simple GET request. The granularity of the data available for each day was impressive, offering insights that could be harnessed in a multitude of ways, from travel planning applications to agricultural platforms that rely on weather forecasting.

Reflecting on the overall functionality, the OpenWeatherMap API offers a robust set of features that can cater to diverse needs. The generous limit of 1,000 free API calls per day presents a valuable opportunity for developers to integrate weather data into their applications without immediate cost concerns. The potential applications of this API are extensive, ranging from integrating weather data into smart home systems to enhancing event planning platforms with weather predictions. The ease of use and the depth of data provided make the OpenWeatherMap API a powerful tool for developers looking to add real-time weather features to their projects.

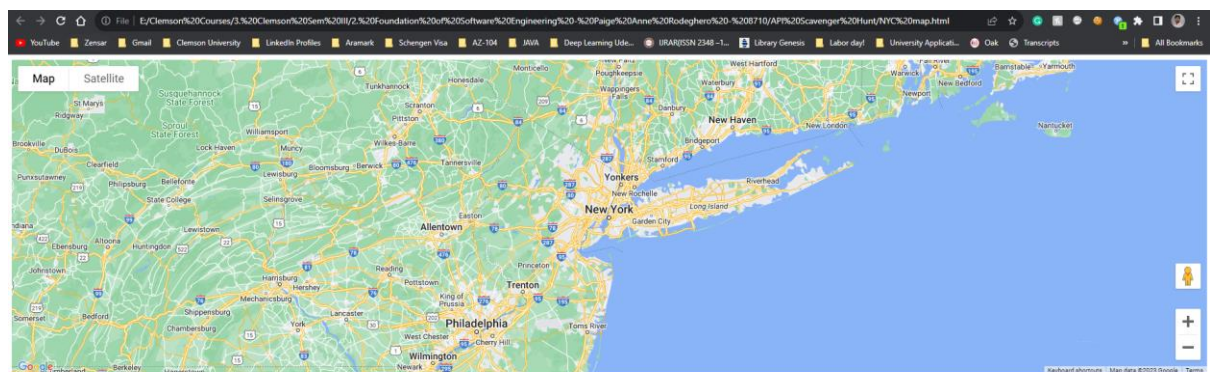
Task2: Solution –

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 </head>
7 <body>
8 <div id="map" style="height: 500px; width: 100%;></div>
9 </body>
10 </html>
```



NYC map.html



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

```
jupyter task2 Last Checkpoint: 4 minutes ago (autosaved)  Logout

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

In [1]: import requests
import json

# actual API key
api_key = 'AIzaSyCMbcQ8k_tazwdjDdhLH0dGK8nZSNV0C9Y'
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.



```
    "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_ejVLCd@Gj@KdBe@z@n1B_@r@I@?dAUPEPGNGLCdBa@Tet@Qd@KNCLCXILEDcROJIJIFC@?B?B?B?B?D@p@DD?@?@?NBF@@?RB@@?D?J@@@D?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><br><b>Oakland</b><br><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><br><b>Central Fwy</b>.",
    "polyline": {
      "points": "e{oeFvqdjVB_@Be@B]K?IS@o@@q@@_@?g@@eB@y@@g@?}@@{@@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><br><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><br><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@  
HWFOFOBMFODQ8KH]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><br><b>Interstate 5 S</b><br><b>Fresno</b><br><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": "krppEvvhUTKPIRIDAFcb@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

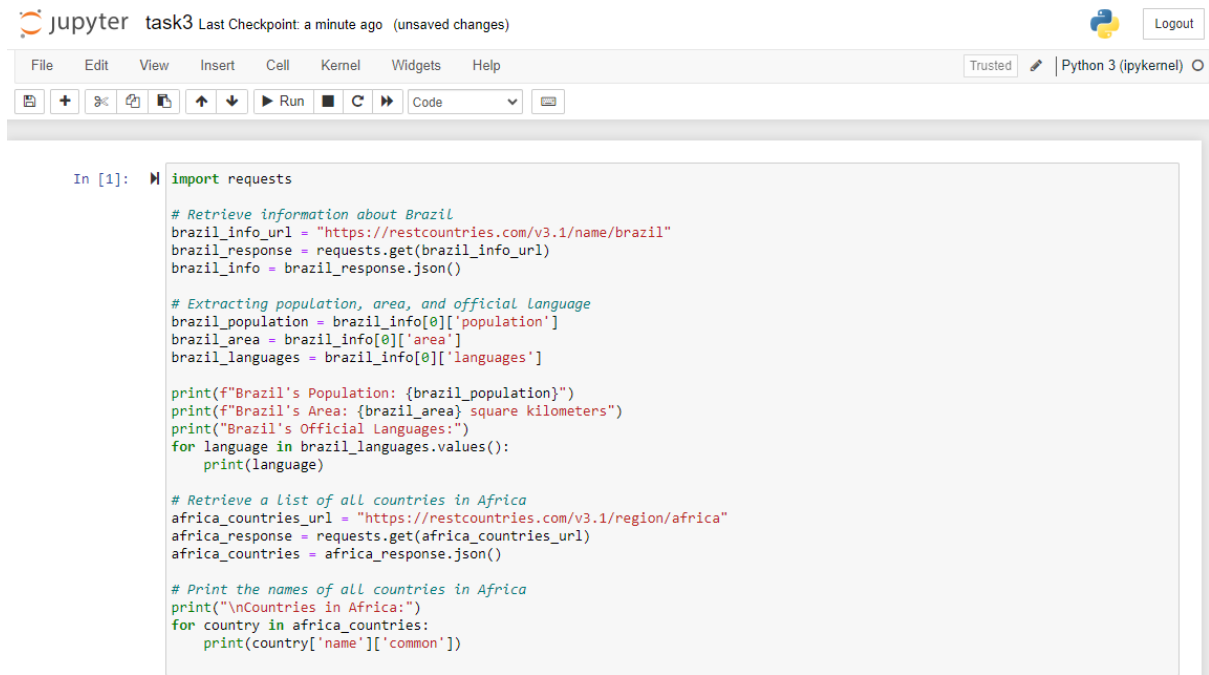
Diving into the Google Maps API was an engaging and educational journey. The process of obtaining an API key was straightforward, thanks to Google's well-documented developer guides. Implementing a map centered on New York City was a matter of a few lines of code, which speaks volumes about the API's user-friendliness. The comprehensive documentation provided by Google made it easy to customize the map's appearance and behaviour to fit the needs of the task at hand. The API's responsiveness and the intuitive interface of the Google Cloud Platform contributed to a smooth development experience.

The task of plotting the shortest route by car from San Francisco to Los Angeles showcased the API's robust capabilities in route optimization. The Directions API, a part of Google Maps services, offered detailed instructions and real-time traffic data, which could be incredibly useful for building navigation tools or logistics software. The ability to integrate such detailed mapping and routing functionalities into applications can transform user experiences in travel planning, delivery services, and more.

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']

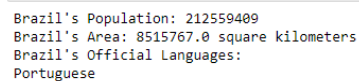
print(f"Brazil's Population: {brazil_population}")
print(f"Brazil's Area: {brazil_area} square kilometers")
print("Brazil's Official Languages:")
for language in brazil_languages.values():
    print(language)

# 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 -



The output of the first code cell is displayed in a light gray box. It shows the following text:

```
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 encounter with the REST Countries API was refreshingly straightforward. 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 exceptionally user-friendly, providing clear guidance on endpoint usage, which made retrieving information about Brazil's demographics and languages a seamless task. The ease of extracting and parsing the data from the API's well-structured JSON response was a testament to its developer-friendly design.

The API's ability to swiftly deliver a list of African countries highlighted its potential for applications requiring regional data segmentation, such as educational platforms or market analysis tools. The simplicity of the API calls belied the depth of information available, offering a rich dataset that could be invaluable for content localization, demographic studies, or even as a foundation for a geography-based educational tool.

In summary, the REST Countries API impressed me with its ease of use and the breadth of information it provides. Its straightforward approach, combined with the comprehensive data, opens up numerous possibilities for developers to integrate a global perspective into their applications. The experience has not only broadened my understanding of RESTful APIs but also sparked ideas for potential applications in various domains.

Task4: Solution –

Source code –

```
jupyter task4 Last Checkpoint: 2 minutes ago (autosaved) Python 3 (ipykernel) 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) Python 3 (ipykernel) 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 proved to be remarkably user-friendly, with well-documented and straightforward endpoints that made integration into my Python script a breeze. The process from obtaining the API key to executing currency conversions was seamless, with clear examples paving the way for a hassle-free experience. The API's response was prompt and the data format was easily manageable, which is crucial for developers looking to implement functionality with minimal complexity.

In terms of application, the API offers essential functionality that could be easily extended to practical scenarios, such as providing real-time currency conversion for e-commerce platforms or financial services. Despite the limitations of the free tier, it serves as an excellent starting point for understanding the integration and use of financial data within applications, highlighting the potential for expansion into more advanced features with its premium plans.