Task1 : Importing libraries

Entrée [10]:

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
import math
import random
import sqlite3
```

Task 2 : Generate random addresses within a specified radius around a central address

1. Initialize Nominatim geocoder

Entrée [5]:

```
geolocator = Nominatim(user_agent="my_geocoder")
```

2. Geocode the central address

Geocoding is the process of converting an address (like "1600 Pennsylvania Avenue NW") into geographic coordinates

Entrée [7]:

```
# Aksing Nominatim api of OpenStreetMap platform <open source> to geocode adr
central_address = "Annecy"
central_location = geolocator.geocode(central_address)
print(central_location)
```

```
Annecy, Haute-Savoie, Auvergne-Rhône-Alpes, France métropolitaine, France
```

3. Description of the process of generating random adresses

To acheieve that, we need guys to follow those steps :

1. Calculate a random distance w within the radius
2. Generate a random angle t
3. Convert polar coordinates to Cartesian coordinates (x and y)
4. Adjust coordinates
5. Reverse geocode to get address

3.1 Calculate a random distance w within the radius

- The Earth is approximately 40,075 km in circumference around the equator .
- One degree of latitude corresponds to approximately 111 kilometers .
- So, to convert kilometers to degrees, we divide the radius in kilometers by 111.0 .

3.2 Calculate a random distance w within the radius

- To ensure that the generated point fall uniformly within the circular area , we use a square root transformation .

$$w = radiusInDegrees * \sqrt[2]{u}$$

- Taking the square root of u ensures that points are distributed evenly within the circle's area .
- Multiplying radius by w gives us a random distance within the radius .

3.2 Generate a random angle t :

- t is a random angle in radians within the range [0, 2π] .
- Multiplying v by 2 * π ensures the angle is uniformly distributed around the circle .

3.3 Generate a random angle t :

- Convert from polar coordinates (distance w, angle t) to Cartesian coordinates (x, y) .
- x represents the horizontal offset from the central point.
- y represents the vertical offset from the central point.

3.4 Adjust coordinates :

- Add the calculated offsets x and y to the latitude and longitude of the central point .
- This adjustment ensures that the generated coordinates are within the circular area centered around the central point .

3.5 Reverse geocode to get address :

- Use the `adjusted latitude` , `longitude` coordinates to perform `reverse geocoding`
- This retrieves the `address corresponding` to the `generated coordinates` using the `Nominatim api geocoder`

3. build a function that launch the process of generating random adresses

Entrée [24]:

```
def find_addresses_within_radius(central_address, radius_km, num_points=10):

    # Initialize Nominatim geocoder
    geolocator = Nominatim(user_agent="my_geocoder")

    # Geocode the central address
    central_location = geolocator.geocode(central_address)
    print(central_location)

    if central_location:
        central_point = (central_location.latitude, central_location.longitude)
        addresses_within_radius = []

        for _ in range(num_points):
            # Generate random coordinates within the radius
            u = random.random()
            v = random.random()

            radius_in_degrees = radius_km / 111.0

            w = radius_in_degrees * (u ** 0.5)
            t = 2 * 3.141592653589793 * v
            x = w * math.cos(t)
            y = w * math.sin(t)

            # Adjust coordinates to be within the circular area
            adjusted_latitude = central_point[0] + y
            adjusted_longitude = central_point[1] + x

            # Reverse geocode to get address
            address = geolocator.reverse((adjusted_latitude, adjusted_longitude))
            addresses_within_radius.append((address.address, adjusted_latitude, adjusted_longitude))

        return [[central_location.address, central_location.longitude, central_location.latitude],addresses_within_radius]
    else:
        return None
```

Task 3 : Test my function

Entrée [26]:

```
central_address = "cusy"
central_address_cordinates = geolocator.geocode(central_address)
radius_km = 5
num_points = 30
addresses_within_radius = find_addresses_within_radius(central_address, radius_km, num_points)

if addresses_within_radius:
    print("Addresses within {} km of {}:".format(radius_km, central_address))
    for address_info in addresses_within_radius[1]:
        print("-------------------------------")
        print("- Address:", address_info[0])
        print("  Latitude:", address_info[1])
        print("  Longitude:", address_info[2])
        print()
        print("-------------------------------")

else:
    print("Central address not found.")
```

```
Cusy, Ancy-le-Franc, Avallon, Yonne, Bourgogne-Franche-Comté, France métropolitaine, 89160, France
Addresses within 5 km of cusy:
-------------------------------
- Address: D 905, Nuits-sur-Armançon, Nuits, Avallon, Yonne, Bourgogne-Franche-Comté, France métropolitaine, 89390, France
  Latitude: 47.74034006017971
  Longitude: 4.201541759761098

-------------------------------
-------------------------------
- Address: Les Craies, Ancy-le-Franc, Avallon, Yonne, Bourgogne-Franche-Comté, France métropolitaine, 89160, France
  Latitude: 47.78257694187887
  Longitude: 4.161753752321661

-------------------------------
-------------------------------
- Address: Le Carreau, Chassignelles, Avallon, Yonne, Bourgogne-Franche-Comté, France métropolitaine, 89160, France
  Latitude: 47.745112486256915
  Longitude: 4.193946654213449
```

Task 4 : Insert generated random adresses into a sqlite database

Entrée [27]:

```
try:
    sqliteConnection = sqlite3.connect('GPSCordinates')
    cursor = sqliteConnection.cursor()
    print("Database Successfully Connected to SQLite !")

    # build the sqlite insert parametrized query
    sqlite_insert_zone_Query = '''insert into Zone (rayon_r, longitude_x_origine, latitude_y_origine, address_origine)
                            values (?, ?, ?, ?);'''

    # build a data tuple
    data_tuple = (radius_km, addresses_within_radius[0][1], addresses_within_radius[0][2], addresses_within_radius[0][0])

    # execute the query
    cursor.execute(sqlite_insert_zone_Query, data_tuple)

    # get the autoincreament zone_id of the last inserted row
    zone_id = cursor.lastrowid

    # commit the resulte
    sqliteConnection.commit()

    # insert generated address into db
    for address_info in addresses_within_radius[1]:
        # build the sqlite insert parametrized query
        sqlite_insert_pointGPS_Query = '''insert into PointGPS (longitude_x, latitude_y, address)
                                values (?, ?, ?);'''

        # build a data tuple
        data_tuple = (address_info[2], address_info[1], address_info[0])

        # execute the query
        cursor.execute(sqlite_insert_pointGPS_Query, data_tuple)

        # get the auto_increament point_gps_id of the last inserted row
        point_gps_id = cursor.lastrowid

        # commit the resulte
        sqliteConnection.commit()

        # build a sqlite insert parametrized query to insert data into `appartient` table
        sqlite_insert_appartient_Query = '''insert into Appartient (idZone, idPoint)
                                values (?, ?);'''

        # build a data tuple
        data_tuple = (zone_id, point_gps_id)

        # execute the query
        cursor.execute(sqlite_insert_appartient_Query, data_tuple)

        # commit the resulte
        sqliteConnection.commit()


    # close the connection
    cursor.close()

except sqlite3.Error as error:
    print("Error while connecting to sqlite", error)

finally:
    if sqliteConnection:
        sqliteConnection.close()
        print("The SQLite connection is closed !")
```

```
Database Successfully Connected to SQLite !
The SQLite connection is closed !
```

Entrée [ ]: