# Workshop 2: Cleaning Tree Locations with BAN API

The tree location data in our database is messy and needs cleaning. Specifically:

- **Arrondissement values** are literal instead of zip codes.
- **Arrondissement values** extend outside of Paris.
- **Address fields** are missing zip codes.

We'll clean the location data using the **BAN API** to retrieve accurate addresses based on the geolocation (latitude, longitude) of each tree.

## Steps Overview

1. Install the PostgreSQL `http` extension.
2. Understand the BAN API and how to make requests.
3. Write a function to retrieve addresses based on geolocation.
4. Parse the returned JSON data.
5. Update the database with formatted addresses.

### Step-by-Step Instructions

#### 1. Install the PostgreSQL `http` Extension

You'll need this extension to make HTTP requests to external APIs.

#### 2. Understand the BAN API

The BAN API provides official addresses based on latitude and longitude.

- **API Endpoint:** `https://api-adresse.data.gouv.fr/reverse/?lon=<longitude>&lat=<latitude>`

- **Example Request:**

```bash
curl "https://api-adresse.data.gouv.fr/reverse/?lon=2.37&lat=48.357"
```

### Example Response

The API returns a JSON object. Below is a sample response:

```javascript
{
  "features": [
    {
      "properties": {
        "banId": "b7f4c615-1356-4e73-87d0-acbe23125963",
        "name": "23 Grande Rue",
        "postcode": "91720",
        "city": "Prunay-sur-Essonne"
      }
    }
  ]
}
```

You will extract the following fields from the response:

- **banId**
- **name** (address)
- **postcode**
- **city**

## 3. Write a Function to Fetch Tree Location

Create a `PL/pgSQL` function that:

- Takes a **tree ID** as input.
- Queries the tree's **longitude and latitude**.
- Constructs the **API request URL** using the longitude and latitude.
- Sends a request to the BAN API.
- Returns the **full JSON response**.

## 4. Parse the JSON Response

Write another function that:

- Takes the **JSON response** from the API.
- Extracts and returns a set of:
  - **banId**
  - **name**
  - **postcode**
  - **city**

### 5. Add New Columns to the `location` Table

Update the `location` table to include:

- `banId`
- `name`
- `postcode`
- `city`

### 6. Write a Function to Update the Table

Create a third function that:

- Takes the extracted address fields (**banId, name, postcode, city**).
- Updates the corresponding columns in the **location** table.

### 7. Combine Everything in One Function

Write a function that:

- Calls the previous three functions sequentially:
    1. Fetch the address using the BAN API.
    2. Parse the JSON response.
    3. Update the location table.

### 8. Test the Function on a Small Dataset

Run the combined function on a small subset of the database to ensure everything works correctly.

---

## Additional Notes

- Remember to use **RAISE NOTICE** for exceptions and errors to help with debugging.
- You can consult the full BAN API documentation [here](here).

--- more verbose version

Here’s a more structured and clear version of your worksheet:

---

## Workshop 2: Cleaning Tree Locations with BAN API

The tree location data in our database is messy and needs cleaning. Specifically:

- **Arrondissement values** are literal instead of zip codes.
- **Arrondissement values** extend outside of Paris.
- **Address fields** are missing zip codes.

We'll clean the location data using the **BAN API** to retrieve accurate addresses based on the geolocation (latitude, longitude) of each tree.

### Steps Overview

1. Install the PostgreSQL `http` extension.
2. Understand the BAN API and how to make requests.
3. Write a function to retrieve addresses based on geolocation.
4. Parse the returned JSON data.
5. Update the database with formatted addresses.

---

## Step-by-Step Instructions

### 1. Install the PostgreSQL `http` Extension

You'll need this extension to make HTTP requests to external APIs.

### 2. Understand the BAN API

The BAN API provides official addresses based on latitude and longitude.

- **API Endpoint:**
  `https://api-adresse.data.gouv.fr/reverse/?lon=<longitude>&lat=<latitude>`

- **Example Request:**

  ```bash
  curl "https://api-adresse.data.gouv.fr/reverse/?lon=2.37&lat=48.357"
  ```

### Example Response

The API returns a JSON object. Below is a sample response:

```javascript
{
  "features": [
    {
      "properties": {
        "banId": "b7f4c615-1356-4e73-87d0-acbe23125963",
        "name": "23 Grande Rue",
        "postcode": "91720",
        "city": "Prunay-sur-Essonne"
      }
    }
  ]
}
```

You will extract the following fields from the response:

- **banId**
- **name** (address)
- **postcode**
- **city**

### 3. Write a Function to Fetch Tree Location

Create a `PL/pgSQL` function that:

- Takes a **tree ID** as input.
- Queries the tree's **longitude and latitude**.
- Constructs the **API request URL** using the longitude and latitude.
- Sends a request to the BAN API.
- Returns the **full JSON response**.

### 4. Parse the JSON Response

Write another function that:

- Takes the **JSON response** from the API.
- Extracts and returns a set of:
  - **banId**
  - **name**
  - **postcode**
  - **city**

### 5. Add New Columns to the `location` Table

Update the `location` table to include:

- `banId`
- `name`
- `postcode`
- `city`

### 6. Write a Function to Update the Table

Create a third function that:

- Takes the extracted address fields (**banId, name, postcode, city**).
- Updates the corresponding columns in the **location** table.

### 7. Combine Everything in One Function

Write a function that:

- Calls the previous three functions sequentially:
  1. Fetch the address using the BAN API.
  2. Parse the JSON response.
  3. Update the location table.

### 8. Test the Function on a Small Dataset

Run the combined function on a small subset of the database to ensure everything works correctly.

---

## Additional Notes

- Remember to use **RAISE NOTICE** for exceptions and errors to help with debugging.
- You can consult the full BAN API documentation here.