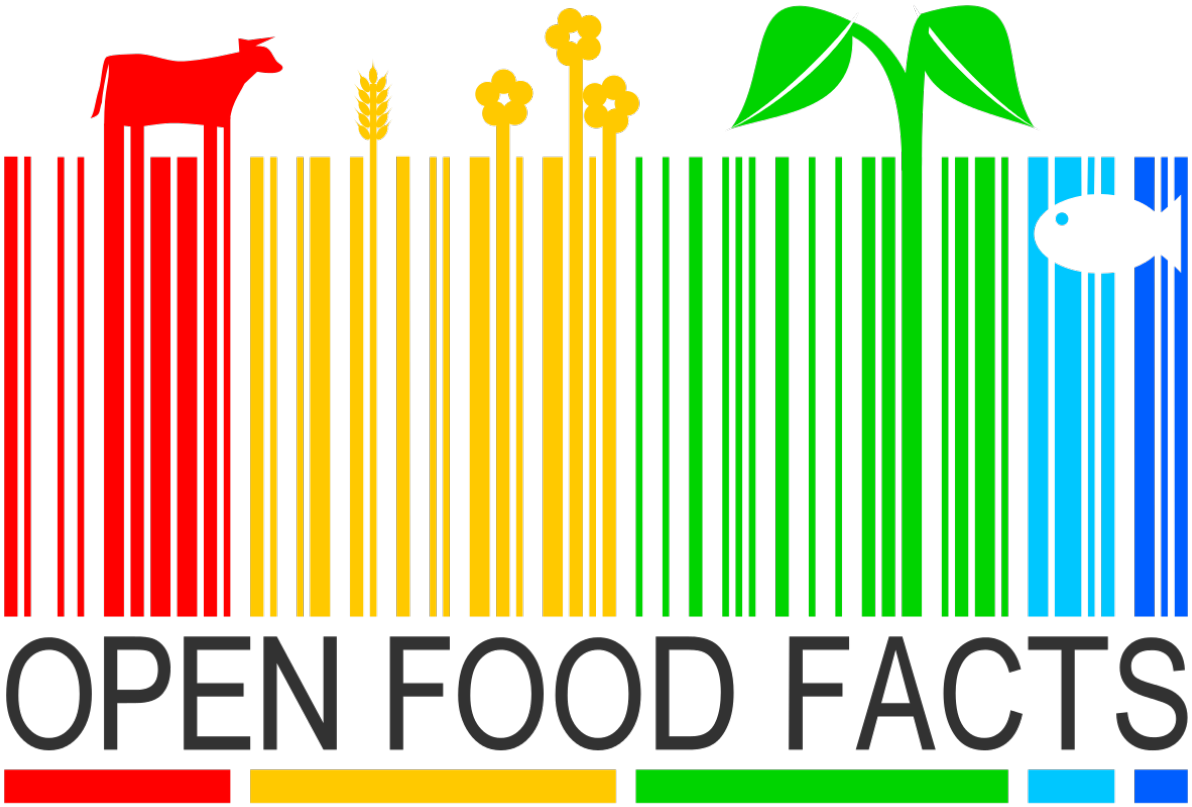
Maxime J.



Projet 5 Openclassrooms : Utilisez les données publiques de l'OpenFoodFacts

Livrables :

- Modèle physique de données (ou modèle relationnel) (pas de dessin à main levée !).
- Script de création de la base de données.
- Code source publié sur Github : https://github.com/Maximedu13/Utilisez-les-donn-es-publiques-de-l-OpenFoodFacts
- Tableau Trello : https://trello.com/b/Vha6EnVx/p5ocuse-openfoodfacts-public-data
- Document texte expliquant la démarche choisie, les difficultés rencontrées et les solutions trouvées et incluant le lien vers votre code source sur Github. Développez notamment le choix de l'algorithme et la méthodologie de projet choisie. Expliquez également les difficultés rencontrées et les solutions trouvées. Le document doit être en format pdf et ne pas excéder 2 pages A4. Il peut être rédigé en anglais ou en français, au choix, mais prenez bien en considération que les fautes d'orthographe et de grammaire seront évaluées !

I. Generalities

The aim of this project is to create a program capable of interacting with OpenFoodFacts public data. The user interacts with the program in the terminal via a search on a MySQL database. The user can therefore consult the products of six categories:

- Oatmeal drink
- Guacamoles
- Corn chip
- Risottos
- Yogurts on fruits bed
- Thick slices of salmon

From the products he selected, he has the means to get a healthier substitute. Then, the user can store it in the database.

This program is hosted on GitHub, here is the link : https://github.com/Maximedu13/Utilisez-les-donn-es-publiques-de-l-OpenFoodFacts

This program is coded in Python and in SQL on Visual Studio Code 1.33.0.

II. Content of the project

The project contains the following programs and files:

- database.py : a program which manages the MySQL connection and assuring that the database open_food_facts in MySQL is created.
- create_bdd.sql : a SQL script that enables tables creating.
- base_action.py : a main program containing two classes : Base_action() and Menu(). The first one deals with the creations, insertions, gets, and deletions of information in database. The second one allows to display them in the terminal.
- queries.py : SQL queries are kept in this file.
- menu.py : file calling the class Menu() and launching the main program.
- README.md : a file containing information about other files in the same folder.
- requirements.txt : containing a list of commands for pip that installs the required versions of dependent packages. The program libraries used are:
  - requests : useful for the database interrogation and the repatriation of Open Food Facts data.
  - PyMySQL : useful for the communication with the MySQL database.
  - pylint : useful for the source code verification and code quality for the Python Programming Language.

III. Approach chosen

It was requested to adopt the Doc Driven Development approach, in other words to, respectively, determine the list of features to provide, write the complete documentation, write the code, check if the doc is well respected, and finally iterate.

I've more or less respected this approach. I used some concepts and tools specific to agile methodologies, but I also integrated some of mines. First, I wrote the steps to follow that I thought I would do for this project. To remind me the tasks to achieve, I created a Trello table, available at this link : https://trello.com/b/Vha6EnVx/p5ocuse-openfoodfacts-public-data. Thus, I divided my program in user stories, and in tasks, but I didn't affect deadlines to my table.

Then, I initialized a Github repository and made my first push. Afterwards, I schematized the Physical Data Model which is a representation of the entities of the relational database. I used the online tool

draw.io to make it. I used it too to make my UML diagrams before. We find in this PDM, three tables (category, product, and favourite), their fields, their primary and foreign keys and thus the different links between these tables. Later, I created the file which managed the connection to the database. From that moment, my program has begun to take shape, and I gradually added the following program features, making me sure that the previous ones worked.

I would say that I adopted an iterative approach.

IV. Algorithms choice

In this part, is developed some of algorithms used in the program.

The reading and script execution of a SQL file algorithm:

The method allowing this is in the program is "execute_Scripts_From_File()" which takes in parameters self, cursor and filename.

First of all, the program attends to open and read the file that we' re interested in via the function open() in Python. Then, we read and close this file. We have to use the function split() to separate all the commands of this SQL file. The delimiter is the semi-colon. From then on, we use a loop "for" to browse through all commands. Inside this loop "for", a try-block attends to via a "if" condition to remove any trailing characters (characters at the end a string). When this condition is checked, we call cursor.execute to execute the command. An except block has responsibility for displaying any error messages if the condition fails.

The insertion of products in database algorithm:

This algorithm is present in the method called insert_products(). First of all, we need to call inside this method the function replace_characters() to get the correct syntax url for the requests categories. For each of these categories, an element is added to the table **list_ch**. Then, we call the method requests.get() and we decode the JSON results (stocked in the variable result) via the method json.loads(). Then, a loop for via a variable "i" will browse from 0 to the length of the results (number of products). Inside this loop for, we assign to each of the products the attributes of the table "Product" presents in the PDM. The id is equal to i + 1. Then a large try-block affects this attributes to the JSON results. There is a special case : it's possible that the product_name_fr in the API is empty, so we need to reassign this variable to the English product name via an if/else construction. We also transform the energy into calories diving the energy by 4.184. Finally, we execute the query and we commit. In the case that a data is missing (by example calories, fat, sugars...) the except-block calls the instruction pass to do nothing, and ignore this product to not insert it.

Recovery of a healthier product algorithm:

To get a healthier substitute, it is sufficient to create a SQL query which selects all the products of the given category which has a nutria "a" score :

find_a_substitute = "SELECT * FROM product WHERE nutri_score='a' ORDER BY RAND () LIMIT 1"

It is possible that there are several substitutes, we therefore need to limit ourselves to only one substitute, that's why it's used LIMIT 1 in the query. It should be a random substitute.

V. Difficulties encountered and solutions found

Many miscoding(errors) were obstacles to the proper functioning of the program:

❌ Error : Python MySQLdb issues (TypeError: %i format: a number is required, not str)

❓ The format string is not really a normal Python format string.

✅ Always use %s for all fields.

❌ Error : #1451 – Cannot delete or update a parent row: a foreign key constraint fails

❓ The foreign key blocks the table deletion.

✅ Replace “ON DELETE RESTRICT ON UPDATE RESTRICT” with
ON DELETE “CASCADE ON UPDATE CASCADE” that applies to the Foreign key.

❌ Error : Python Error NameError: name '...' is not defined

❓ This very variable is inaccessible.

✅ Use the keyword global to define it.

❌ Circular dependencies.

❓ It happens if two modules import each other.

✅ Move the path import to the end of the node module or combine the files.

VI. Possible improvements

To improve the project, it would have been possible to use these libraries: termcolor and/or colorama to give color to the program to use within the terminal.

I could also develop this program via a graphical interface, as Tkinter, for more readability.