Animal Map

1. Introduction

Animals are all over the planet, from enormous to microscopic, dangerous to harmless, rare to common. There are in fact so many of them that we cannot count every single one. Some estimates that there are approximately 20 quintillion individuals, which can be also written as 20 000 000 000 000 000 000. This is such a considerable number, that we cannot process it clearly in our brain. But we found a way to bring some order in that mess, and this thing is to classify them.

While classifying such a large number of animals may seem useless for some, it allows us, humans, a better understanding of our history, and relationships and connections between all living things. Understanding those aspects of the animals' evolutions and characteristics makes it easier to study varied species, which can bring innovative solutions to current problems or make unexpected links between some animals come to light.

There are a lot of possibilities in classifying animals, we could range them simply by color, size, or even rareness. But for a more complete classification, we chose to do it by physical characteristics, such as, the body covering, body shape, appendages, and many more.

But even after classifying them, the number of discovered species is still growing fast. While some go extinct, others evolve into new species. As humans love to classify things, we must store all known data into different sections, from extinct animals to newly discovered ones, increased space is needed. Furthermore, the substantial number of species makes the visualization of a single species in a specific location complicated. And often, the data is stored in large chunks which makes intricate details harder to read and comprehend. Those problems often discourage motivated people that are willing to search for and learn more about animals in specific areas. Such information should be easy to access and comprehend, for a specialist as for a novice.

The aim of this project is to classify animals, in a more visualizable way. Classified species will appear on a world map in the form of a single marker. Each marker can be clicked on for a popup to appear and display different information about the said species.

For this paper, we chose to simplify things, and classify animals in tables of their common name. In those tables, animals will be differentiated by their species. Then by using simple queries, make changes to the database on the user's demand.

The contributions of our work are as follows:

- We analyze the array of actions possibly useful for the user as to manipulate the database.
- We design and implements function such as creation, deletion, modification into the program
- We send the added information to the database, so it can be implemented into the map
- We demonstrate the well-functioning of the program by displaying the map.

The rest of this paper is organized as follows: Section 2 describes the motivations. Section 3 presents the design and implementation of the proposed schemes. Section 4 concludes this paper.

2. Motivations

The main motivations for this project were, by default, the interest that we have in animals. They are essential for this planet, and we could not imagine a world without them. Making this idea a real project may help people, by granting them the possibility of a better understanding of the animals that surround us, via easier access to various information. Although other projects of this type already exist, and probably have a better classification than ours. Nonetheless, we wanted to try our best to create a simple yet efficient program to give people an easier time while learning about animal species and their characteristics.

3. Design and implementation

We chose to write the program using the **Python** language on **Jupyter Notebook**. **Python** being very popular, and an easy and accessible language, we hope that the program could be a source of inspiration to some other people wanting to create a more solid and maybe complete program.

To achieve an easy and understandable program for any user, we aim to use simple questions and queries. To do this, we propose when the program is launched, a list of all possible actions to the user. This list includes 8 different actions. The different actions go from displaying the map, to terminating the program. Furthermore, the map includes a legend, giving the meaning behind the different colors of the markers.

3.1 Design

First, we chose to introduce into the program 2 base **.csv** files, with each of them containing multiple members of a group known under a common name. Those files will be automatically read at the start of the program, placing their markers into the map. The data contained in those files is not part of the database manipulation part that this program proposes. Thus, they cannot be modified directly in the program. The part of interest to us, include 6 of the 8 actions proposed to the user. The rest are simply the "View Map", "Exit Program" options. To manipulate the database file, we chose to use the python library **sqlite3**. For an easier understanding, we will refer to **sqlite3** as **sql**.

Each table has the same formations, meaning they all contain the same column number and names for the classification. Every species is therefore classed with the same columns. Those columns are, "Name" which we use as a primary key because of its uniqueness. "Family", the family of the said species, "Discovery Year" which represents the year of discovery, "Endangerment Level" containing the color that the placed marker will have, representing its endangerment based on the map legend. "Latitude" and "Longitude", as indicated are the coordinates where the species can be found. Note that those coordinates are of course not the exact one where a species is found, but rather where it could be observed, and/or has been found in the past. And finally, "Wiki" in which we store the name of the species that is displayed on Wikipedia. This name is most of the time, the scientific name of the species with a '_' between words. This allows us to easily create a link which leads directly to the species page if clicked on while pressing ctrl.

3.1.1 Get access to the database

In each of the actions interacting with the database, there are primordial steps to follow. The first one is to create a connection to the database, which is done by using the **sql** command *sql.connect('database_file.db') and* storing it into a variable that we will name connection. Then initializing a cursor using the previous variable with *connection.cursor()* and storing this in another variable named **cur**. Using *cur.execute('query')* lets us execute any sort of action we would need to handle the database. For each action, a different query is set. After the query being executed, we must commit the changes into the database. For this, we need to use the *connection.commit()* command. All that is left to do is now to close the cursor, and end the connection with the database using *cur.close()* and *connection.close().*

All the changes being made have now been committed to the database, and the cursor and connection have been closed. Closing the cursor after its use is important, it frees resources associated to the query. Same goes for the connection, closing it will free the resources of the database.

3.1.2 Get the names of all tables

In some of the actions, we need to go through all tables, which creates a problem, how can we get all the tables currently in the database with a simple query. Since the user can

freely add or delete tables, neither the name nor the number of tables stays the same during the execution of the program. To resolve this problem, we first connect to the database using the previously shared steps (section 3.1.1). We then execute a specific query, *"SELECT name FROM sqlite_master WHERE type='table';"*. This query returns to us the names of all tables, with which we can print all of them using *print(cur.fetchall())*. We can also use a more intricate code to fetch only one name at a time, to be able to use them in specific functions. For the more intricate part, we first initialize a variable *row* that we will set to *cur.fetchone()*. Then we create a while loop using *while row is not None:*, and add inside of it *function_name(' '.join(row))* and *row = cur.fetchone()*. This will run the function for every existing table in the database. We must use the *' '.join(row) because* the retrieved names from fetchone(), are not on a string form. The above used *sqlite_master*, is an internal table that is present in all sqlite databases, and its content describes the database's schema. That is why using this returns us all the currently existing table names.

3.1.3 Update the map

To update the map after every action, we implemented a function named UpdateMap. This function takes for argument the name of the table that will be updated. It starts using the primordial steps (section 3.1.1). It will then send a query "*SELECT * FROM "+ table_name*. This query will be read using **pandas** library, *df=pandas.read_sql_query(query, connection)*, and then transfer the data into a **.csv** file with *df.to_csv('anything.csv').* Note that the file's name we write inside the parenthesis is of no importance. With the data now in a **.csv** file, we can read it using **pandas**, *data = pandas.read_csv('anything.csv').* the file being now stored into a variable, we must get its information by using a *for* loop with *data.iterrows()*. The implementations of the data into markers placed on the map and creating popups for each of them just uses some **html**, and other things not related to the database itself. It is why we will ignore them in this paper.

3.2 Implementation

As previously said, there are 6 actions that directly manage the database. Those are respectively, "Show Data", "Add Data", "Delete Data", "Modify Data", "Create Table" and "Delete Table".

Show Data aims to print on the terminal, all the species contained in one table. It will be using the previously explained steps to create a connection to the database (section 3.1.1), and then ask for the user from which table does he wants the data to be displayed. For this, the names of the existing tables are printed using the *cur.fetchall()* part (section 3.1.2). We then add the input of the user into a query *"SELECT * FROM " + input.* Next, by using the library **pandas**, we read the query and using the opened connection to the database, apply the query using *df=pandas.read_sql_query(query, connection)*, and *print(df.head())* for printing the results into the terminal. The results will take the form of an array.

Add Data is a function whose goal is to add a new species to the database. It will start in the same way as Show Data, connection to the database, cursor (section 3.1.1), and then will ask in which table does the user ants to add the new data. It will do so by using the *sqlite_master* system (section 3.1.2). Afterwards, the user is asked to input all the information on the new species. Each of those individual statements will be put inside a single query, *"INSERT INTO "+table_name+" VALUES (?, ?, ?, ?, ?, ?, ?)".* followed by its execution *cur.execute(query, (name, family, …)).* Each of the '?' refers to an input typed by the user, they are set as variables of any type that will be defined into the execution command. We put each input inside the parenthesis in the order they have been created in the table. After committing the changes to the database, we use the UpdateMap function (section 3.1.3), and close everything that needs to be closed.

Delete Data is a function to delete a species by entering its name. It uses the same steps as the other function (section 3.1.1) and follows the same path as *sqlite_master* (section 3.1.2). The differences lie inside of the *while* loop. We set inside of it, a query deleting the row containing the input name of the species using the previously seen *' '.join(row)* (section 3.1.2), and putting it inside of the query *"DELETE FROM "+' '.join(row)+" WHERE Name = (?)"*. The rest does not change from other actions.

Modify Data is used to modify information about an existing species without having to delete and recreate one. Note that only one piece of information can be changed at a time, the user must recall the action if more columns need to be changed. The function starts like the other ones (section 3.1.1), then the user must input the name of the species and the column he wants to modify data from. The function then uses *sqlite_master* (section 3.1.2) to search the wanted species inside the existing tables. The new data is asked to the user, and a query is then created, *"UPDATE "+table_name+" SET "+Column+" = "+new_data+" WHERE Name = ?"*. After its execution, the map is updated (section 3.1.3), and the cursor and connection are closed.

Create Table is a function to as the name indicates, create a new table. Nevertheless, the table is not completely new, the only new characteristic it has is its name. It connects itself to the database (section 3.1.1), asks the user the name of the new table he wishes to create, and simply creates it using a long query containing all the characteristics of the tables included in the program. The query goes as follow*, 'CREATE TABLE ' +name+ '("Name" TEXT NOT NULL UNIQUE,"Family" TEXT,"Discovery Year" INTEGER,"Endangerment Level" TEXT,"Latitude" NUMERIC,"Longitude" NUMERIC,"Wiki" TEXT,PRIMARY KEY("Name"));'* We decided to include the full query into this paper as it resumes the tables in every details. The next steps are like the other functions, execution, commit, closing.

Delete Table is our last function, and it is there to completely delete a table, without having to delete every single species contained in one. It is a rather simple function. It works like all the other ones, connecting itself into the database (section 3.1.1) and using *sqlite_master* (section 3.1.2) to display the names of the current existing tables. The user then must input the name of the table he wants to delete, and by using the query, *"DROP TABLE "+name*, the table is deleted entirely. The function ends normally.

4. Evaluation

While the program has its advantages, it also has disadvantages in some domains. The improvements are seen through a comparison with a standard database program that stores animal data. The data in our program is way clearer, and easy to read. When looking at a certain country on the world map, you can simply look at the markers in an area, which seem more intuitive than a large block of rudimentary text containing endless information. The experience of documenting species of an area is more pleasant than in a more classic data storage approach. From that point of view, the program improves the user-friendly side of data storage.

Meanwhile the outside aspect can seem more appealing, the program being new, and from a non-experimented student, it lacks in robustness and stableness. A single miss typed word in some of its functions can conclude in some error interrupting the program. There is no option to add more columns into the database or add random comments into the popup. Or about the option of adding an image of the said species inside the popup, only working for the base **.csv** files and not for the database one.

4.1 Comparison with another program

We can compare our program with another project that has done the exact thing, the website Map of Life. On this website, we can see that instead of single markers representing a single species, a species is represented by large colored zones on the map. While this can be more representative of the distribution of a species throughout an area, having more than one of these zones can create overlaps which will render the map unreadable. And so, instead of having one map containing all the species, they chose to have a single map for every single species. This allows way more information to be displayed for the user but makes the switch from one species to another more complicated, which would make our program more competent in that manner. But to answer this problem, Map of Life also has an option, which allows the user to click on a country, and then list all known species of animals living in this area in distinct categories. After clicking on a name of a species, a popup appears displaying a description about the said species. With all of this in mind, our program is completely defeated in an informative way.

Although the website has a better way to display information, our program still has an advantage over the possibility to personally modify the data. Indeed, the map being a website with rights reserved, it does not let any user modify the data. Giving this permission to random users will probably cause problems with the trustfulness of the information. So, our project, being open source and the database being more focused on individual user uses, and so only connected to one device, still has the upper hand in a small but existing domain, the private usage of the map.

5. Conclusion

While the current results of the program are satisfying, you can guess that the program has still a long way to go if the project wants to enlarge itself, and truly attain the set goal. A program allowing any user to create, search, and study animal species throughout a world map. All the previously stated and explained actions work as wanted, which let any user willing to try the program out, a large array of manipulations regarding the map and the database. However, we also encountered limitations throughout the creation of the program. Indeed, the **python** library used to create and modify the world map (**folium**) does not provide the possibility of deleting a created marker. So even if the database deleted a species, or modified its location, the old marker will remain on the map. After some research, there is a conceivable way for marker deletion. This would be done using layer creation and deletion but can only be done using the **java** programming language. We decided not to include this possibility in the current state of the project.

As seen in the comparison with Map of Life (section 4.1), there are out there more sophisticated and thoughtful programs aiming at the same objective. With this in mind, we can conclude that even if our project has not attained its final goal, people wanting to learn more about animals can do so on already existing platforms. This can be still a satisfying outcome, as our project had for its main purpose to open solutions to knowledge thirsty individuals.