ICS4U
Cristina Zhou
Ms.Woodford
June 4, 2018

Report

My final project is option 2, automated database. My task is making a program that will initialize, read in, and manipulate a database of cars and accessories. My program also has to be able to perform the following jobs: add car to inventory, remove car from inventory, find car in inventory, find accessory that matches with given car model number, display databases, output inventory, and quit.

Python is a perfect language choice for this project. First, python has been used in quantitative and qualitative analysis for a very long time, especially in data science and finance field, which are fields that dealing with large databases daily in essence. Fortunately, python is very friendly towards database programming, it supports various forms of databases like MySQL. It is fast, efficient, clean -- and most important, portable. Its APIs are portable with various databases, which makes applications written in python easilier to migrant and port to other interfaces. Moreover, there are many build-in modules that works perfect with databases. For instance, SQLite, peewee and json. I was trying to write my program that links with SQLite databases when I first saw my project requirements: it's lightweight, easy to use, doesn't require extra installation or configuration, and stores in a single disk file -- which is to say, we can start to work as soon as import the sqlite3 module from the python standard library. However, I started to see its incompatibility with my project as I continued to work on. First of all, I need to create at least two separate databases for my car and accessory inventory before I have to attach them to my third file that specially write to operate them. Secondly, it is not user-friendly. My goal is to design a program that can interact with the user inputs through terminal or console, and it is difficult for me to directly interact with user through SQLite3. My second try was with peewee, a simple and small Object-relational Mapping, it supports various databases and has numerous extensions modules available. Unfortunately, I encountered some issues and have to give up halfway. Later I did some research on Django and flask as well, however, they are all overly complicated and unnecessary for my current program. My final try is with json after almost three weeks died in vain. What caught my attention about this JavaScript Object Notation is its readability and Python's support to it -- Python has a build-in package called json for encoding and decoding JSON data (Lafaro, 2008).

I was contemplating the structure of my program while I was still exploring the syntax of json. My hypothesis was there will be at least three files. One is a file that has databases which contains preset test data, one is the core file that does all the functions, and one json file that has output inventory data. My first step is to create a folder and then create a db_creator python file which will initialize the preset databases, and a card.py file that will do the major work. I imported json and tabulate module (to formatted output as pretty tables). Afterwards, I created the Car class, Accessory class, CarDBEngine class, and the main function if __name__ == '__main__'. Interestingly, it may sounds deviant but what I found that particularly works for me is working on the main function first, so I wrote the code that will display the menu and went back to the top.

Under the Car class, I scratched some empty functions. __init__, which is used to initialize the instance of my Car class. What my __init__() function does is to initialize the property of my Car class, which is why I put self, model_num, year, color, maker, model, body_type, quantity in its brackets. Noticeably, self is a crucial variable here that can save myself a lot of work. As we know, a class can have multiple objects, and the self variable can tell the method clearly which object has called it.

I then create a list_cars_info(cars) method, which will display the car database in a print table form that is formatted by the tabulate module; and a add_cars() method that will ask users for inputs and append values to the inventory. In addition, I put two @staticmathod one line above each method respectively since they could be used as standalone functions but somehow I still them to be kept in Car class.

Move to the Accessory class. I did almost exactly same with the Car class, and basically all I need to do is to change some variable names.

The CarDBEngine(object) class is the most vital one here. I started with initialization, assigning dictionaries to self.car_db and self.accessory. It is a necessary step for me to search for a specific item. After that I create a find_car_by_model_number method which would searches a given model number in the keys of the car_db dictionary and lists the item if been found. Then I created a add_car method and a remove_car_by_model_number method. The logic behind the remove function is similar to the find car function. As for the add_car method, it writes data to the json file and append the newly added car to the car_db dictionary. Next I create two functions that will display car and accessory inventory, in other words, methods that will display the information of cars that stores in previous list_cars_info and list_accessories_info methods. Quit function is easy to work on as well. Unfortunately, I encountered some issues when working on my export_inventory and find_matched_cars functions, and soon I discovered that my add_car method and quit function were problematic as well. I did a lot research online and found that my problem laid on json serialization. To simply put, data have to be in text form when exchanging between the browser and server, so I have to find a way to convert python objects to json and vice versa. I ended up put two tojson() method under the Car and Accessory class respectively, and loads json file when retrieving data back.

I turned back to work on my db_createror.py file, imported json module, as well as car and accessory class from carb.py. I then made two example databases that contains some accessories and cars as test data. Eventually I use a bunch of if else statements to determine the result of user inputs.

In conclusion, the difficulty of this project does not laying on mathematically or computationally but more conceptually. We have to understand how small things work under the big picture and be aware of the logic and structure of the program we are writing, therefore, modularization and flow diagrams are rather helpful in general.