Rapport IN104

Martine GRANGÉ
Erika MONNIN

23 Mai 2019

1 Introduction

The project is to program a flashcard system using object oriented programming (OOP). First, we needed to understand how a flashcard system works to know what type of object we would need on python. Then, we needed to learn how to use Git, which is a new tool for us. Finally, we had to separate our work in 3 steps : coding the management system, coding the training system, and coding the graphical user interface.

2 Management system

First of all, what objects do we need ? It was clear that we needed a class "Card", to define data about our cards, and a class "Deck", containing a lot of methods to easily arrange and re-arrange the deck. But because we were not familiar with OOP, we first created a complete programm with different functions in order to arrange the deck. But it was muddled. So we created a complete Deck class, with (what we thought were) adapted attributes and methods.

In fact we had too many useless attributes, such as "Length" which we don't need because the information is already given in the list containing the cards (len(list)), or "Position" in the list. For this later point, at the first thought, we see it as essential to delete cards with just a pop() instead of an entire loop. However it would have been complicated to manage cards thanks to their position, because everytime a card is added in the deck, all the positions change. So pointing to cards thanks to the identifier remained the best solution. A usefull object to know when to review cards was the *datetime*. It was not easy to understand how the mathematical operations worked on it, but once we enquire about it we decided to use its "delta" method, thus to only work with periods and not with dates.

At first we had troubles with github because we were using the website to share our code, but everything could have been done through the terminal. At the second lesson we started over with git on our computers not to need the website anymore. And it was much more easier afterall ! We only needed to be careful not to change things in the same programm at the same time, otherwise it became impossible to commit.

At this time we still needed to implement the management system. Some parts were easy to code, some on the other hand were tricky.

For the "Add a card to the deck" function, we first did not verified if the card the user wanted to add was already in the deck or not. This way the deck could contain several cards with the same identifier ! Which is a huge problem to identify the cards ! We then added a loop to be sure that a card with the same identifier does not already exist in the deck.

For the "Load a deck" function, understanding how pickles file can be loaded was the first part so we could easily load our decks. Moreover we thought it was the first thing a user should do when he or she wants to manage a deck. This is why the first question the user should answer is : "do you want to load a deck ?". Then we had troubles with the saving of decks : we wanted to save decks in the pickle file while only lists can be saved, because a class cannot be called. Another problem was that the deck was not saved after we added cards into it. To know if the deck was correctly saved we decided to print the number of cards in the deck at every moment. This way , if it reads "O cards" we would know the deck was not correclty saved. We also added a checker of the deck after the user quits the management system, to be sure the deck is saved even after the computer is off.

For the "Edit a card" function, creating many "if" conditions, one for each attribute that can be changed, was too long. So we put the attribute as an input of the function, so that the method of the card would be used this way : card.attribute=change. But it was not as easy as we thought : the method "setitem" was needed in our cardclass to be able to do it. To check if the card has been well edited, the "Print a card" function was created. One point remained : verifying that the attribute put as an input exists in our card class, and sending an error message if not. Furthermore, some of the attributes are strings, others numbers. In Python 2, two different inputs are used : a simple input for numbers, a raw-input for strings. So we implemented it in three loops, one for figures, the second one for strings, and the default one is an error message informing what kind of attribute can be changed.

Finally, we added the "upper()" method to all our strings so that the computer doesn't make a difference between "identifier" and "IDENTIFIER", which would have been a problem in the functions depending on the input of the user. We also added a test condition to be sure that the user doesn't ask things the program can't do. If he does, the program asks the user to try again.

At the very end, to clear our files, we decided to change the name of the file containing the class card. But it was saved in every card of the pickle file. Each single card had to be rewritten with the new name to ensure the deck will be recognized by the programm.

3 Training system

The first important part was the reflection about the list of cards to be reviewed. We thougth about different ways to code it :

- writing the date of the last time the card has been reviewed as an attribute of the class card, and then everytime the user wants to train, the algorithm calculates the period from this date to today, and see if it's more or less than the time needed for the learning.

- calculating, before the saving, the date from which the card needs to be reviewed, and then if today is after this date, the user reviews the card.

We finally decided to mix the two solutions. So we added a "review" attribute to class card, and we created a function aiming to calculate the date from which the card needed to be reviewed, and to compare it to today's date. If the card needs to be reviewed, it is added to the list of the day cards.

Testing functions helped us to ensure the algorithms ran. But the code didn't work because of what we wanted to test : if the function works with a wrong type argument. In this case, the testing function didn't worked because the function could not be called with a wrong type argument.

When finally the training system worked, the shape of the answers given by the user could be slightly different to what we waited for. So we decided to write only the first letter of the answer on the backside of our cards, and to have a multichoice answer. This way the user only needs to type one letter and we met again the form of the management system, an easy one. We by the way added a choice "Q - Quit" to let the user decide when he wants to stop training. If the user quits before the end of the session, what has been learned is automatically saved, so that the user can resume at any moment.

Finally we enhanced our training system with some questions, some statistics about the learning, some figures... We wanted it to be clear for the user, readable even through the terminal. It is why a "timesleep" has been added to let the user the time to see its results before going to the next question.

4 Graphic interface system

Looking back at what we did, the hard part really was the implementation of the GUI!

The first thing was to understand what enables *Tkinter*. The tutorials were not really helpfull for what we wanted to do, because it did not explained us how to open different windows when clicking on buttons. We only found how to create a window class and add methods. But then, if we created a new window, the things we wanted to have on the second window (like an Entry) appeared only on the first window, because it was the root to which we applied the methods.

We tried to work without defining everything in the class but having a main code apart from it, and defining only the important methods. But this way our windows did not read anything.

Finally we found examples on internet of how to manage different frames, with the possibility to go backwards to the firsts ones. We only needed to apply it to our

systems.

For the management system it was rather easy. In fact at the beginning we had troubles with the indentation, but once we made the indentation clear it worked perfectly. We adapted our raw code to Tkinter. We needed to beware of the little exceptions, such as what to do if the user clicks on the button but he did not enter anything in the entry line, or if the card the user wants to change is not in the deck, or if the deck is empty... We used the package "tkinter.messagebox" to print different exception message when such things occured. At the end we even changed the colors and the fonts to make it more attractive to the user.

On the other hand, we faced more difficulties when it came to the training system. The main difficulty was the creation of the loop of the training. Should we create it in an apart frame, or in a method of the main class. Implementing the loop in a separated class allows us to avoid destroying all attributes. But it supposed to use global variables.

Two different ones were used : my-deck, from class Deck, and a-revoir, a list.

The first step in our training system is to load a deck. We used the same algorithme as in the management system, and we thought it would be the same : once the deck is loaded, the attributes of the global variable my-deck is changed so we can use it elsewhere. In fact there was a problem, because once we went to the next window, my-deck was empty. What happened with the loading is still a mystery ! Why would it work in the management system but not in the training ?

So we changed our way of doing things by using functions and not methods. But a problem appeard with a-revoir. To work, our system uses a-revoir[0], but before the deck is loaded a-revoir = [ ] so we are out of range. When we added a "if" condition to skip the problem, we only had white windows opening... Global variables can only be changed in the initialization of the class.

5 Conclusion

As a conclusion, this project enabled us to discover how to use classes and git system, manage inputs from the terminal and draw the graphic user interface up. The quite autonomous lessons compelled us to seek information on the internet as we will do as engineers. As the final algorithms works as expected, the satisfaction is great. And as these flashcards will be useful on the campus, it is even greater.