Project06 : Chat bot assembly – Report

Seyed Ali Firooz Abadi

04/15/2022

Requirement

Create a chat bot that could handle all queries. The project was split into 6 sub-projects:

1. Extract data from the district
2. process it (extracted data) based on questions
3. make content available in a command-line interface
4. handle any user query
5. report statistics on interaction of a session, across sessions
6. Full Chat bot presentation

And the chat bot was required to answer at least 11 specific queries. It was expected from us to professionally handle advanced programming topics, such as: exceptions and file operations.

Specification

The path that I took was to use Python programming language to do the entire project. Below I will briefly explain each project and how it was handled.

Project 01: extracting data

To solve this problem, I used 3 libraries: Requests, BeautifulSoup, and OS. I started the program by asking the user to enter a number for the district the user wants to check. The user's input will then go in a while loop to check if it is the right input. I used try except to account for any unforeseen errors, and gave the user a maximum of 3 tries after which the program will terminate itself.

If the user input was 78, the rest of the program will run, but if not, it will show a message informing the user that other districts are not supported yet.

If in fact the user input was 78, we proceed to request the web page HTML file using the Request library and the get() function. Then we parse the HTML code using first "lxml" and then "html5lib", we do this so that we could have a better HTML code since each parsing method is slightly different (lxml is better in handling bad files and html5lib is better organized).

After getting a good HTML file of the district web page, we start by extracting data from it using BeautifulSoup functions. Different filters (queries) were used to find the data (due to the bad structure of the HTML file (where CSS styling elements were implemented in line rather as classes!))

After successfully targeting the HTML tags, I used a BS function to get the text inside that tag and save it as a string. Some elements were harder to target and I was left out with a list of strings rather than just one string.

String manipulation were then done on some data that needed more attention.7. After getting all needed data as strings stored in there own variables, I started the process of opening the output file using the “with” keyboard to eliminate the need to close the text file afterwards.

I have written all the variables inside the text file with a simple formatting to make it easier to read.

After finishing the writing process, I was ready to apply my user-defined function to count the words, lines, characters, and empty spaces and output the statistics to a different text file.

The user-defined function used the OS library to strip the formatted text file.

First I used a for loop to manipulate the strings and count the words and characters. Each time the loop runs will increase the lines, words, char, and spaces count.

Project 2: processing data

Since I have already processed the data in the first assignment and each relevant information was saved in the relevant variable, I have created a dictionary and pickled it (professor said this would have extra credit) to the 2nd assignment. In the second assignment: 1. Created a class named “Person” that would hold all the district representor information (name, phone, …) a. To do that, I created a method to de-pickle the data received from the first assignment.

b. extra : for future reusability, I created a method that would de-pickle a file if a path (string) was given to the constructor.

c. In the constructor, I used the received dictionary from the de-pickling process, and extracted it to relevant variables (firstName, phoneNum,...)

d. I have also created a method to access the dictionary based on an index (not a key). (This will be later used in next class)

e. Finally, sice all the attributes are private, I created access methods. I have used python “@property” to make all the accessors have the ability to be called like an attribute.

2. Created a second class (file) called “ChatManager” where we will teach our bot to look for words inside the user questions. a. Defined 4 functions called “toString_*** ” to create a string from list based on a separator (space, tap, next line, numbered list), these function havethe ability to include a custom string (passed to the method) when processing the list. (see implementation in the main code)

b. Defined a method to ask the user for input and check if it was “exit” or “quit” and then return the user input as a string if it was not any of the

above mentioned words.

c. Finally used a method called “answerQuestions” to process the user input and return relevant information if it found anything. Note that this method is as good as the training process behind it. Right now it is very basic and will only give relevant information if it finds these statements:

(full name, home address, columbia address, phone number, political party, county, personal information, committee assignments, service time)

Also, if you want information about me, just ask something that contain “your dad”.

3. Finally I just used some attributes from the “Person” class to show that I have processed the data correctly and also initiated the chat bot , so now you can enter any question you want.

Project3: developing command-line interface

I developed an abstract class that would be the base for creating classes that can handle user’s questions. All above mentioned classes follow the same name pattern that would have “… _info” in them. Those classes will hold to lists and a method, the first list called “training list” is responsible for holding key words that the method will use regular expression to check against user’s query. The second list “training list str” should be holding names (or explanations) that would be offered to the user upon matching the query with the right training list element.

Note that the training list is actually a 3 dimension list, and needs to be like that even if 3 dimensions were not needed.

I included a pickled object from the 2nd project and created a method to include it into my 3rd project, to successfully do that, I needed to create the same class “OO_Person” again but only with it’s attributes so that the pickled object (district 78) could be successfully depickled. Finally I created a chat manager to combine all the functionality of the different classes. It will depickle the district object and answer questions.

Project 4: handling user query

I created a class to make use of the FuzzyWuzzy library which is based on the Levenshtein library.
The methods in the class would take in a question and a multidimensional list, then it would iterate through the inner lists one by one until it gets to the strings. After that it will check each string in the list with the question and if the end result ratio is higher than the limit, it will return True value to another list.

After going through all the sub lists, we will end up with a new list that would have the same size of our original training list but with True and False values. To make this easier to use, I added another method to go through the new boolean list and give the indexes of all the True values.

I have also created a uniqueAns method to make sure that I only got one match. In case of more than one match, the method will check if they are under the same group (like if both are in the “name” group, like first name and last name), which in that case, the method will also return True. In case I got more than one match from different groups, the method will just return False.

To successfully implement that, I replaced my last attempt (to use regular expressions) from Assignment 3, where I had the user question split into words and cross check with predefined keywords using Regular Expression. I replaced the RE method with the newly created method and everything was the same.
I did add one extra feature for the “personal information” in such a way that it someone would ask

about the “personal information” and then about something relevant, the app would go and match the specific information in the personal information with the user question.

Project 5: reporting statistics

I created a class called logger and changed the behavior of the “print()” method to write to console and also to output to an external text file. This way I could log all the print method inside the chat bot, but still faced the problem of not getting the “input()” method logged to the file.

To solve the aforementioned problem, I used a simple trick to first get the prompt message to the logger class as an attribute called “question” and then running the “loggedInput()”, it requested the information from the user with a simple “input()” and then logged the user input to the log file and returned the user input back.

I add an extra private method called “writeTime()” to add time stamps in the logged file if needed for later. “flush()” method just needed to be implemented to stop the program from crashing.

Finally, before exiting the program, I called the method “writeCSV()” to log a CSV file with the statistics of that session.

Development highlights

Implementation

1. I used an abstract class to be the blueprint for creating classes that are intended to answer questions.

2. From there, I ended up with 5 classes inheriting from the abstract class that would have 5 type of information to answer (basic_info, personal_info, other_info, log_info, bot_info)

3. I have also created 3 helping classes, that would assist in presenting and matching the information and be used in other classes as composition (OO_personal_info, query_matching, to_string)

4. I, then, used composition again to put everything together, by creating the chat manager (chat_manager) class

5. To create logs, I created a class called “chat_logger”

6. Finally, in the main code file, I called the chat logger first and then started the chat bot.

Testing

1. I started by testing all the required queries, then tried to include miss-spelled words. I ended this part by also eliminating some words and trying to optimize the chat bot performance

2. I went then into testing all other functionality such as exit, or entering wrong district numbers

3. I then tried to run the bot for a very long time to see if it would crash

4. Finally I tried to corrupt some aspects of the file structure that I had to test the chat bot ability to handle such situation.

Problems and Lessons

1. Some times it is better to parse an HTML through different libraries (lxml is better in handling bad files and html5lib is better organized)

2. Pickling dictionaries, lists, etc is an easy way to transfer data between different projects.

3. Although it is possible to pickle and depickle a class object with all it's attribute, it should be noted that both projects (the one pickling and the one depickling) should contain the object's class to successfully use the object methods and attributes across projects.

4. You should be mindful when depickling an untrusted file because it could change the system and currupt it. This problem can be solved by using a cryptographic signature, which can be implemented using the HMAC module.

5. It's very tricky to log files to an external file and in the same time print them. To do that we need to minupilate the default print method to first log and then print, and same goes for the input method.

Reuse

To easily reuse others' code, I created a method in project 2 to import and create a person object if it was represented with a dictionary in certain order. This way reuse will not be limited to just python, but also other languages as well.

Future work

- Use regular expression in the first project to find relevant information

- find some other way to calculate the number of answers so that you don't add extra lines in the main method to make the count right.

- use regular expression to search for keywords in the person's personal information and return it to the user, then go for the fuzzy search if no keyword was found.

- change the size of "__createBoolList" to match the size of "aList" exactly. This way will save memory.

- create method to read the data out a text file and populate a dictionary to make use of others code