Python project
My first chatBot

Instructions & general information :

⇒ This project is to be done exclusively in Python.

⇒ Project attachments :
- Presidents' speeches files

⇒ Team organization :
- This project is to be carried out in pairs (**only one trinomial is allowed** if an **odd number** of students).
- The list of teams is to be given to teachers at the end of the first project follow-up session at the **latest.**

⇒ Key dates :
- Publication date: **04/11/2023**
- Follow-up session 1: Week of **13/11/2023**
- First filing date: **11/26/2023** at **11:59 p.m.**
- Follow-up session 2: Week of **27/11/2023**
- Final deposit date: **12/19/2023** at **10:20 a.m.**
- Date of defense: Week of **18/12/2023**

⇒ Final deposit:
- The project will be submitted using Moodle
- Project code containing **.py** and **.txt** files
- The **.pdf** report
- A **README.txt** file listing the programs and how to use them in practice. This file must contain all the instructions necessary for execution; it is very important to explain to the user how to use the tool.

⇒ Evaluation
- Detailed scale: to be provided later
- Final project grade = Code grade + Report grade + Defense grade
- Reminder: project grade = 20% of "Python Programming" course grade
- Members of the same team can receive different marks depending on the effort they put into the project.

⇒ **Collaborative working**
- Project team work will be tracked via Git
- The Git tutorial will be online the week of **06/11/2023**

Code organization :

⇒ The rating of the code will mainly take into account :
- Implementing requested functionalities: moving forward as best you can, but NOT off-topic
- The quality of the code supplied: organization into functions, **comments**, meaningful variable names, respect for file names.
- Ergonomics: Ease of use of the user interface

Teaching concepts covered :

⇒ Completing this project will help you apply the following pedagogical concepts:
- Basic concepts, 1D lists, 2D lists, functions, files and collections

⇒ To help you realize this project, you can rely on :
- TI101 course materials (I, B, P)
- Books, various courses on the web. **ALERT PLAGIAT!!** it's not about copying entire programs.
- **Efrei** teachers during project follow-up sessions

Description

This project is about text analysis. This project will enable you to understand some basic concepts used in text processing and help you understand one of the methods used to develop chatbots and/or generative artificial intelligences such as chatGpt.

Obviously, we're not talking here about manipulating neural networks, but in this project we're going to focus on a method based on the number of occurrences of words to generate intelligent answers from a corpus of texts. The aim is to design a system that can answer questions based on the frequency of words in the corpus.

This application is based on the following basic algorithm:

Data pre-processing: your program begins by collecting and pre-processing a set of documents in order to understand the nature of their contents, before using them to prepare answers. This phase cleans up the text by removing punctuation, converting letters to lower case, and dividing the text into words (or "tokens").

Creating a TF-IDF matrix: For each unique word in the documents, you'll need to calculate a TF-IDF vector using the TF-IDF method. Each word is associated with a vector whose dimension is equal to the number of documents in the corpus. This creates a TF-IDF matrix where each row represents a word and each column represents a document.

Question representation: When a question is asked, the chatbot performs the same pre-processing on the question. It then calculates a TF-IDF vector for this question, using the same vocabulary as the documents. The question vector has the same dimension as the vectors associated with the words in the corpus.

Similarity calculation: The chatbot calculates the similarity between the question vector and the word vectors in the corpus, using cosine similarity or another similarity measure. This enables it to determine which words in the corpus are most similar to the question.

Selecting the best answer: The chatbot identifies the words in the corpus most similar to the question, based on their TF-IDF similarity score. It then selects the answer that contains the greatest number of these similar words.

Provide answer: The chatbot returns the selected answer as the answer to the question asked.

Required work

To guide you in the realization of this application, we will work it in 3 parts:

1. **Part I**: Development of basic functions for understanding the content of the files supplied.
2. **Part II**: Calculation of the similarity matrix and generation of automatic responses.
3. **Part III**: Generalizing the application to cover various themes.

Part I

File database

In this project, 8 text files are provided in a directory called "speeches". These files represent speeches by 6 French presidents at their inaugurations.

The name of each file is in the following format: **Nomination_[president's name][number].txt** where :

- **[name of a president]**: Is one of the following names: Chirac, Giscrad d'Estaing, Mitterrand, Macron, Sarkozy
- **[number]**: Is an optional field representing a sequential number. It appears in the names of files where the same president has given several speeches.

Basic functions

In order to analyze the contents of these files, we first ask you to develop the following functions:

- Extract the names of the presidents from the names of the text files provided ;
- Associate a first name with each president;
- Display the list of chairmen's names (watch out for duplicates) ;
- Convert the texts in the 8 files to lower case and store the contents in new files. The new files should be stored in a new folder called "cleaned". This folder should be located in the main directory of the ***main.py*** program, at the same level as the "speeches" directory.
- For each file stored in the "cleaned" directory, browse its text and remove any punctuation characters. The final result should be a file with words separated by spaces. Please note that some characters, such as the apostrophe (') or the dash (-), require special treatment to avoid concatenating two words (e.g. "herself" should become "elle même" and not "ellemême"). Changes made at this stage should be stored in the same files in the "cleaned" directory.

Notes :

1. These functions can be implemented using the predefined modules covered in the course, as well as the "bone" module.
2. The Python code used to browse the list of files with a given extension and in a given directory is as follows:

```python
import os

def list_of_files(directory, extension):
        files_names = []
        for filename in os.listdir(directory):
                if filename.endswith(extension):
                        files_names.append(filename)
        return files_names
```

```
 # Call of the function
directory = "./speeches"
files_names = list_of_files(directory, "txt")
print_list(files_names)
```

The TF-IDF method

In word processing, it is often useful to represent words as numerical vectors to facilitate information retrieval, document classification and/or text analysis.

One frequency-based method for generating numerical vectors for words is called **TF-IDF** (Term Frequency-Inverse Document Frequency). Here's how it works:

TF (Term Frequency): The first part of the **TF-IDF** measures how often a term (a word in our case) appears in a specific document. The more frequently a term appears in a document, the higher its **TF** score for that document➔ For each word in each file, you are asked to count the number of times that word appears. To do this, we need at least one function that calculates the number of occurrences of each word in a text.

- Write a function that takes a string of characters as a parameter and returns a dictionary associating with each word the number of times it appears in the string.

IDF (Inverse Document Frequency) : The second part of the **TF-IDF** measures the importance of a term in the entire corpus of documents. Terms that are very frequent in many documents will have a lower **IDF** score, while terms that are rare will have a higher **IDF** score. ➔ For each word, calculate the **decimal logarithm** of the inverse of the proportion of documents in the corpus that contain that word. This gives more weight to rare words and reduces the weight of common words.

There are several formulas for calculating the IDF score. For the sake of simplicity, the formula to be used here is the following (without the +1):

$$IDF\ (word) = \ log_{10}\ (\frac{Total\ number\ of\ documents\ in\ the\ corpus}{Number\ of\ documents\ in\ the\ corpus\ containing\ the\ word})$$

- Write a function that takes the directory where all the files in the corpus are located as a parameter and returns a dictionary associating the **IDF** score with each word.

Note:

It is permitted to use the "math" module to call its predefined function calculating the logarithm (**log10**).

Final score: The **TF-IDF** score of a word in a document is obtained by multiplying the **TF** score by the **IDF** score.

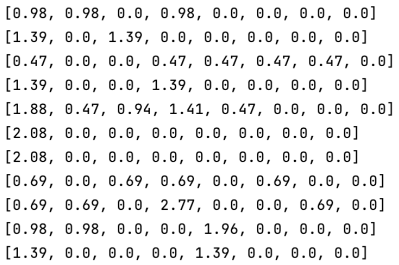
- TF-IDF = TF * IDF.

Thus, the **TF-IDF** score of a word in a given document is a numerical vector that reflects both the frequency of the word in that document and its relative importance in relation to the corpus as a whole.

To generate numerical vectors for words, each word in the document corpus is assigned a vector of **TF-IDF** scores. These numerical vectors represent the distribution of each word throughout the corpus. Together, these vectors form a matrix called a "**TF-IDF matrix**", where each row represents a word and each column represents a document.

- Write a function that takes the directory where the files to be analyzed are located as a parameter and returns at least the TF-IDF matrix.
- The number of rows in the resulting matrix must be equal to the number of unique words in all the files, and the number of columns must be equal to the number of files in the directory➔ If necessary, write a function that calculates the transpose of a matrix (**Warning!!** do not use a predefined function).

Example of a part of the TF-IDF matrix:



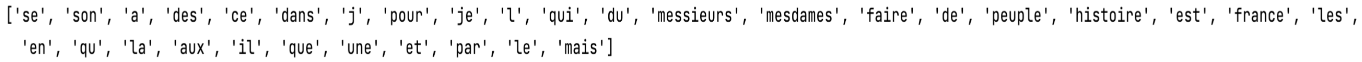
```
[0.98, 0.98, 0.0, 0.98, 0.0, 0.0, 0.0, 0.0]
[1.39, 0.0, 1.39, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.47, 0.0, 0.0, 0.47, 0.47, 0.47, 0.47, 0.0]
[1.39, 0.0, 0.0, 1.39, 0.0, 0.0, 0.0, 0.0]
[1.88, 0.47, 0.94, 1.41, 0.47, 0.0, 0.0, 0.0]
[2.08, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[2.08, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.69, 0.0, 0.69, 0.69, 0.0, 0.69, 0.0, 0.0]
[0.69, 0.69, 0.0, 2.77, 0.0, 0.0, 0.69, 0.0]
[0.98, 0.98, 0.0, 0.0, 1.96, 0.0, 0.0, 0.0]
[1.39, 0.0, 0.0, 0.0, 1.39, 0.0, 0.0, 0.0]
```

Features to be developed

Based on the above functions, write programs that allow you to :

1. Display the list of least important words in the document corpus. A word is said to be unimportant if its **TD-IDF** = 0 in all files.

   Example:



```
['se', 'son', 'a', 'des', 'ce', 'dans', 'j', 'pour', 'je', 'l', 'qui', 'du', 'messieurs', 'mesdames', 'faire', 'de', 'peuple', 'histoire', 'est', 'france', 'les',
  'en', 'qu', 'la', 'aux', 'il', 'que', 'une', 'et', 'par', 'le', 'mais']
```

2. Display the word(s) with the highest **TD-IDF** score
3. Indicate the word(s) most repeated by President Chirac, excluding "unimportant" words.
4. Indicate the name(s) of the president(s) who spoke of the "Nation" and the one who repeated it the most times.
5. Indicate the name(s) of the president(s) who spoke about climate and/or ecology
6. ~~Aside from the so-called "unimportant" words, which word(s) did all the chairmen mention?~~

Main program

The aim here is to write a temporary main program in which you are asked to :

- Have a menu
- Access previous functions according to user request

Don't forget that using Git is mandatory. All project teams must have a repository on Github, and code must be shared with all team members.

Part II

Having processed the documents in the corpus and generated the TF-IDF matrix, we now turn to the automatic generation of answers to the questions.

To do this, we must :

- First, calculate the TF-IDF of the question asked;
- Search the corpus for the most relevant document(s), i.e. the one containing the most words similar to the words in the question➔ It would probably contain more answers to the question asked;
- Generate the answer based on the selected document.

Let's get started!

1. Question tokenization :

Tokenize the question into individual words in the same way as for the corpus documents. This involves dividing the question into words and removing punctuation, deleting capital letters, and any additional processing carried out on the texts of the corpus documents.

- Write a function that takes the text of the question as a parameter, and returns the list of words that make up the question.

2. Search for the question words in the Corpus :

- Write a function that identifies the terms in the question that are also present in the document corpus. Ignore terms absent from the corpus, as they will have no associated TF-IDF values. In other words, look for terms that form the intersection between the set of words in the corpus and the set of words in the question.

3. Calculate the TF-IDF vector for the terms in question :

In this section, the TF-IDF matrix of the corpus must have N rows and M columns, where N = number of documents in the corpus (8 in our case) and M = number of words in the corpus (1681 in our case).

> Those who have represented the TF-IDF matrix in the form of a dictionary must adapt to obtain an M-dimensional vector associated with each document.

The TF-IDF vector in the question must be of dimension = M (same number of columns as the TF-IDF matrix). It is calculated as follows:

- Associate a TF (Term Frequency) score with each word in the question, equal to the frequency of each word in the question➔ Set a 0 for words in the corpus that are not part of the question.

  Example:

  If a word appears twice in a question consisting of a total of 10 words, then the TF for that word is 2.

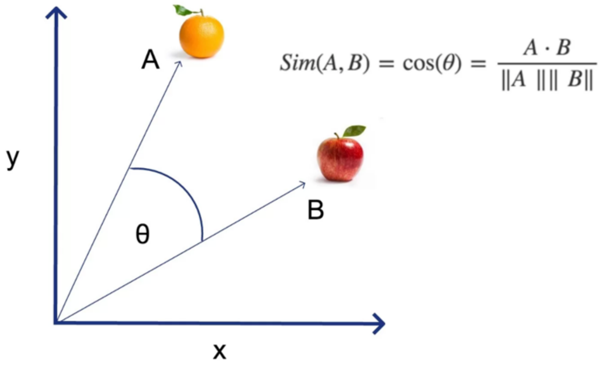
- The IDF scores of the words in the question are the same as those of the same words in the corpus. So all we need to do is use these previously calculated IDF scores to calculate the TF-IDF of the question terms.

  ➔ The python function to be written here must return the TF-IDF vector of the question.

4. Calculating similarity :

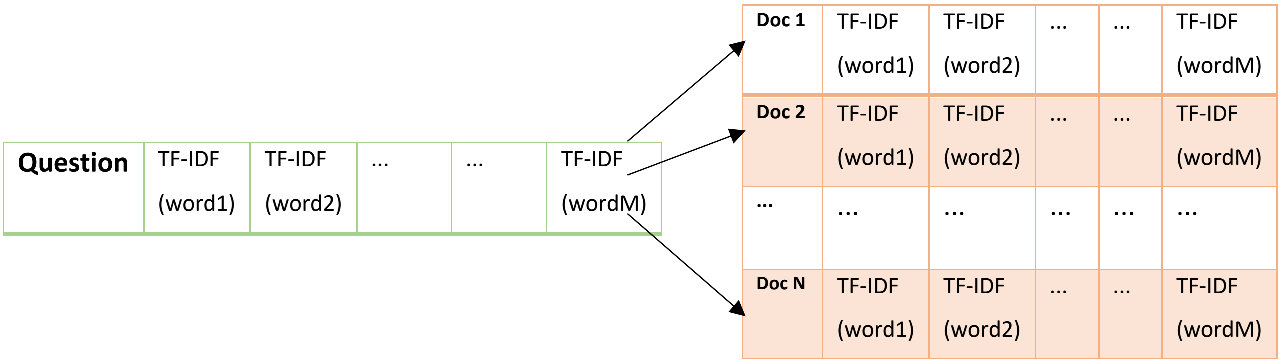
Now that the question vector has been generated, we need to find out which document in the corpus it is most similar to. To do this, we'll use cosine similarity. This is a measure used to evaluate how similar two vectors are in a vector space. In particular, it is often used in natural language processing to measure the similarity between two texts represented as vectors (for example, TF-IDF vectors).

Mathematically, cosine similarity measures the cosine of the angle between two vectors A and B in a multidimensional space. The smaller the angle, the higher the value of its cosine, and therefore the higher the similarity between the two vectors.



$$Sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

In this project, we are asked to calculate the similarity of the question vector with each of the N vectors (rows) of the TF-IDF matrix, and to designate the document with which the question obtains the highest similarity value.



Question	TF-IDF (word1)	TF-IDF (word2)	...	...	TF-IDF (wordM)

Doc 1	TF-IDF (word1)	TF-IDF (word2)	...	...	TF-IDF (wordM)
Doc 2	TF-IDF (word1)	TF-IDF (word2)	...	...	TF-IDF (wordM)
...	...	...	...	...	...
Doc N	TF-IDF (word1)	TF-IDF (word2)	...	...	TF-IDF (wordM)

To calculate the cosine similarity between two vectors A and B without having to manipulate the angle separating the two vectors, use the following formula:

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\cdot\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

In Python, this requires the implementation of the following functions:

a. The scalar product (dot function) :

This is a function that takes as parameters two vectors A and B of the same dimension M (number of words in the corpus) and calculates and returns A . B = $\sum_{i=1}^{M} A_i B_i$

b. The norm of a vector :

This is the length of a vector. The function takes a vector A as parameter, then calculates and returns the square root of the sum of the squares of its components:

||A||= $\sqrt{\sum_{i=1}^{M} A_i^2}$

NB: The **sqrt** function in the math package may be used here.

c. Calculating similarity :

This is a function that takes two vectors A and B as parameters and returns the result of the following score: $\frac{Scalar\ product\ of\ A\ and\ B}{Norm\ of\ the\ vector\ A\ X\ Norm\ of\ the\ vector\ B}$

5. Calculating the most relevant document

To find the most relevant document name, simply implement a function that takes as parameters the TF-IDF matrix of the corpus, the TF-IDF vector of the question and the list of file names in the corpus. It must calculate the similarity of the question vector with each of the document vectors, then return the document name corresponding to the highest similarity value.

At this level, the name of the file to be returned will be the one contained in the "*./cleaned*" directory, so we need to provide a function that supplies its equivalent in the "*./speeches*" directory.

6. Generating a response

Generating an answer automatically is a complex process, involving the use of advanced methods for managing semantics in text. The aim here is to do it with the simplest of methods. Nevertheless, any improvement to the method described in this document is welcome, provided it does not call on predefined functions or word-processing modules.

The method requested here is as follows:

- In the TF-IDF vector of the question, locate the word with the highest TF-IDF score and return it.

Explanation:

A term with a high TF-IDF is a term that on the one hand has a high TF, which means that the term appears frequently in a specific text compared to other terms in that same text. This suggests a potential importance of the term for understanding the content of that text (which is the question in our case here). On the other hand, it also implies that the term is rare elsewhere in the corpus. That is, the term is specific to only this text.

In text analysis, a term with a high TF-IDF can often be considered characteristic or informative for a particular text. This may indicate keywords, specific concepts or significant elements that differentiate this text from others.

- In the relevant document returned in step 5, locate the first occurrence of this word and return the sentence containing it as the answer. A sentence is defined here as the text surrounded by two ".

Example:

Question: `" Peux-tu me dire comment une nation peut-elle prendre soin du climat ?"`

Relevant document returned: `investiture_Macron_duplicate.txt`

Word with the highest TF-IDF: `climat`

The response generated: `et je songe bien sûr à françois hollande, faisant oeuvre de précurseur avec l'accord de paris sur le climat et protégeant les français dans un monde frappé par le terrorisme.`

7. Refine an answer

To bring more soul to the generated response and remove its crude appearance, it is possible to make improvements, such as imposing a capital letter at the beginning of a sentence and a "." at the end.

One suggestion is to set up responses according to the form of the question asked. To do this, create a dictionary of possible text forms, with associated model responses. This dictionary can also be enriched with polite formulas.

```
# Non-exhaustive list of suggestions
question_starters = {
    "Comment": "Après analyse, ",
    "Pourquoi": "Car, ",
    "Peux-tu": "Oui, bien sûr!"
}
```

Examples:

Question 1 : "Peux-tu me dire comment une nation peut-elle prendre soin du climat ?"

Answer 1 : Oui, bien sûr! et je songe bien sûr à françois hollande, faisant oeuvre de précurseur avec l'accord de paris sur le climat et protégeant les français dans un monde frappé par le terrorisme.

Question 2 : "Comment une nation peut-elle prendre soin du climat ?"

Answer 2 : Après analyse, et je songe bien sûr à françois hollande, faisant oeuvre de précurseur avec l'accord de paris sur le climat et protégeant les français dans un monde frappé par le terrorisme.

Main program

The final main program must have a menu offering the user two options:

- Access Part I functionalities at the user's request
- Access Chatbot mode, allowing the user to ask a question

Part III: Bonus

The functionalities described in Parts I and II are basic, and describe only part of the complex process of text analysis.

If you want to go further, you can make a number of improvements, as long as they don't involve predefined functions, because the primary aim here is to demonstrate skills in analyzing situations and designing appropriate solutions:

1. In the text clean-up phase, in addition to deleting capital letters and punctuation, it is also possible to remove accents, replace characters such as « l' » by « le » or « la » , « qu' » by « qui » or « que », etc.

2. Processing conjugated verbs
3. Enrich the corpus with similar documents or documents dealing with the same subjects (climate, nation, ...) and observe :
    a. Impact on the quality of the response generated
    b. Impact on program runtime
4. Develop the tool so that it can answer thematic questions. To do this, we will:
    a. Consider uploading files dealing with other themes. For example, create an application that answers questions linked to one of the 17 objectives (of your choice) of the UNESCO (United Nations Educational, Scientific and Cultural Organization) sustainable development goals.

1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

    b. Modify the menu to allow users to choose their theme.
    c.

Appendix :

Here are a few references from which you can download documents in text format.

- https://www.gutenberg.org/ Vast selection of free books, public domains and classic works in text format;
- https://commoncrawl.org/ Data set containing texts extracted from the web;
- https://www.kaggle.com/datasets data sets for various competitions and projects, including textual data sets;
- https://dumps.wikimedia.org/ Wikipedia Dumps available for textual analysis of Wikipedia articles.