VEEMbot : The Intelligent Chatbot

TUTORED PROJECT DEVELOPER GUIDE
Master 2 Internet of Things

Enguerrand BOITEL
Elio ELHOYECK
Mohamad MAKHLOUF
Vincent RAMOUSSET

Tutor : Raphaël COUTURIER

2020 - 2021

Table of contents

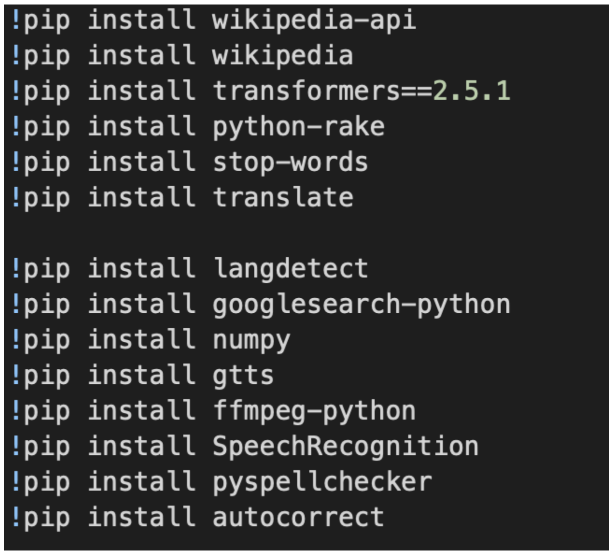
1 Packages used

1.1 Installations

As we used Google Colab, we have some packages which are already installed on the system like Keras, TensorFlow or MatPlotLib. VEEMbot requires some other packages that are not already installed on Google Colab.

The packages we have used are (Figure 1) :

- *"wikipedia"* and *"wikipedia-api"* : For the requests on the Wikipedia site,
- *"transformers"* (Version 2.5.1) : For the open-ended questions,
- *"python-rake"* and *"stopwords"* : To retrieve the keyword in the sentence,
- *"translate"* and *"langdetect"* : For the translation of the sentences and to detect the language,
- *"googlesearch-python"* : To retrieve more information about the sentence,
- *"numpy"* is used for the sentiment classification,
- *"gtts"*, *"ffmped-python"* and *"SpeechRecognition"* : To get the audio from the device of the user, to transform the text to speech and the speech to text,
- *"pyspellchecker"* and *"autocorrect"* : To correct the question asked by the user.

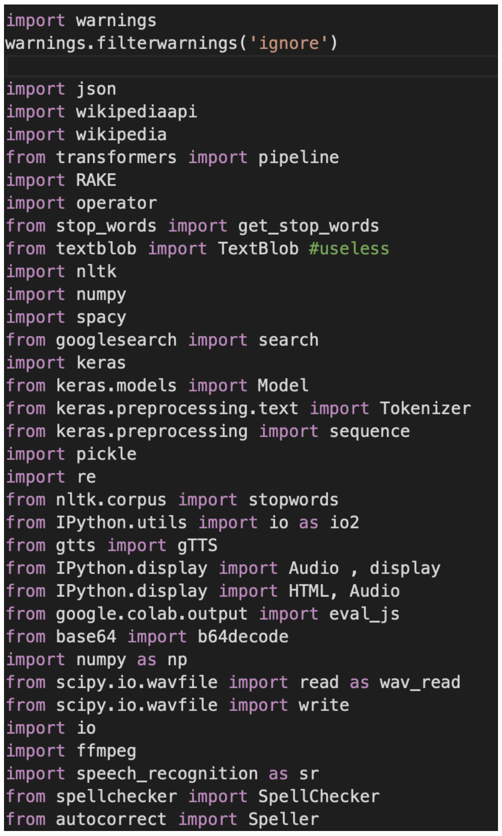


```
!pip install wikipedia-api
!pip install wikipedia
!pip install transformers==2.5.1
!pip install python-rake
!pip install stop-words
!pip install translate

!pip install langdetect
!pip install googlesearch-python
!pip install numpy
!pip install gtts
!pip install ffmpeg-python
!pip install SpeechRecognition
!pip install pyspellchecker
!pip install autocorrect
```

Figure 1: Installation of the packages used by VEEMbot

1.2 Imports

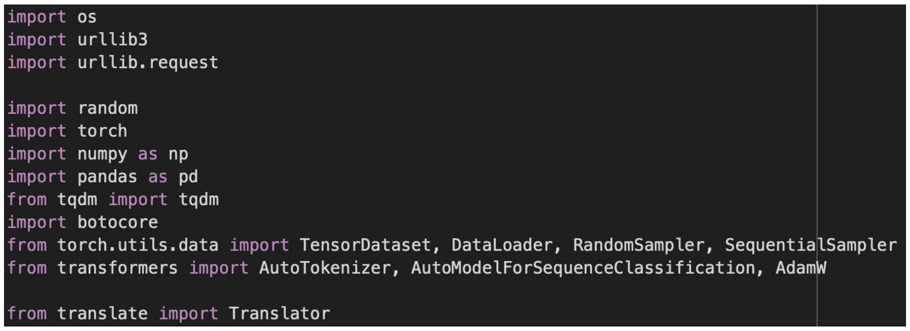
After the installation of the packages for the opened-ended questions and all other features we will see, you can import them by using these lines (Figure 2) :



```python
import warnings
warnings.filterwarnings('ignore')

import json
import wikipediaapi
import wikipedia
from transformers import pipeline
import RAKE
import operator
from stop_words import get_stop_words
from textblob import TextBlob #useless
import nltk
import numpy
import spacy
from googlesearch import search
import keras
from keras.models import Model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
import pickle
import re
from nltk.corpus import stopwords
from IPython.utils import io as io2
from gtts import gTTS
from IPython.display import Audio , display
from IPython.display import HTML, Audio
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
from scipy.io.wavfile import read as wav_read
from scipy.io.wavfile import write
import io
import ffmpeg
import speech_recognition as sr
from spellchecker import SpellChecker
from autocorrect import Speller
```

Figure 2: First part of the importation of the packages used by VEEMbot

And for the closed-ended questions (Figure 3):



```python
import os
import urllib3
import urllib.request

import random
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import botocore
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW

from translate import Translator
```

Figure 3: Last part of the importation of the packages used by VEEMbot

2 Main functions

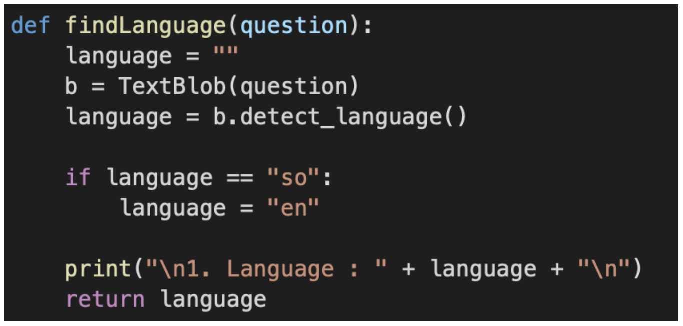
2.1 Analyze sentence

2.1.1 Get language

To get the language of the question asked by the user, we have tested different packages.

The first package we have used is *"langdetect"*. *"langdetect"* can be used to detect the languages of a question, but sometimes, when the text is not very long, the language detected is not the good language. For example, with this package, if we asked *"Is Montbéliard in France ?"*, the language detected cannot be the good language. Here, the good language is *English* but it returns *French*.

So, we have decided to used another package which is *"TextBlob"* (Figure 4). *"TextBlob"* is more accurate than *"langdetect"*. On all the tests we have done, all the languages returned by the function were all in the right languages.



```python
def findLanguage(question):
    language = ""
    b = TextBlob(question)
    language = b.detect_language()

    if language == "so":
        language = "en"

    print("\n1. Language : " + language + "\n")
    return language
```

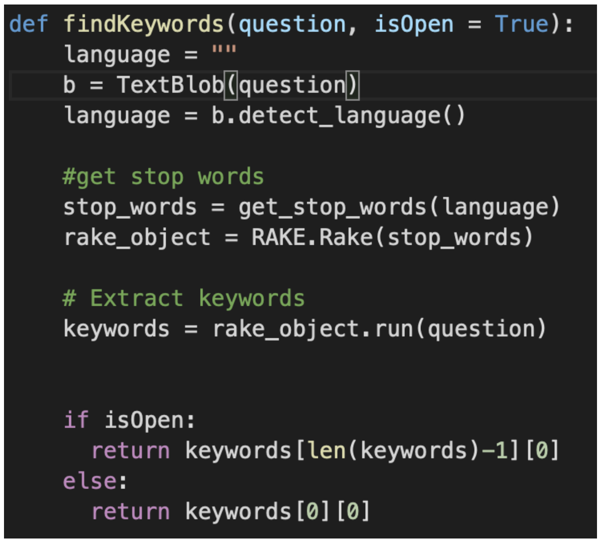
Figure 4: Function to detect the language of the request

2.1.2 Get keyword

To find the keywords in the sentence, we have used two principal packages : *"rake"* and *"stop_words"*.

The *"stop_words"* package helps us to get all the possible words that are so common that is unnecessary to index them or use them in our search of the keyword. The possible words can be, for example : *the*, *a*, *on*, *of*, ... We are using this package because it gives to us the stop_words in different languages such as *English*, *French*, ...

The second package *"rake"* gives us the all possible words that can be important in the sentence. For the open-ended question, the best keyword is the last keyword in the table returned by *"rake"*. While, for the closed-ended question, the best keyword is the first one in the table (Figure 5).



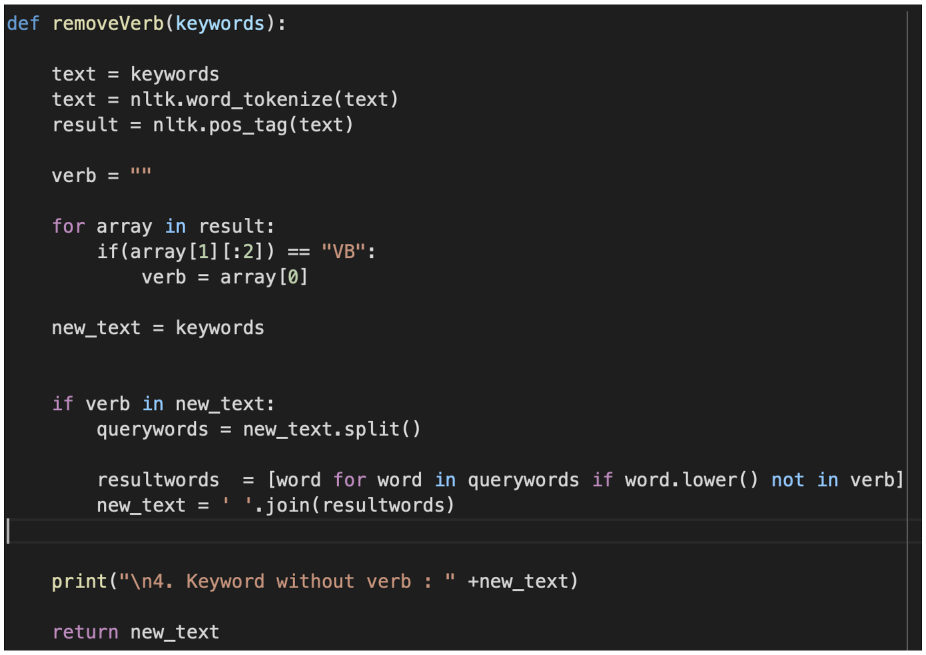
```python
def findKeywords(question, isOpen = True):
    language = ""
    b = TextBlob(question)
    language = b.detect_language()

    #get stop words
    stop_words = get_stop_words(language)
    rake_object = RAKE.Rake(stop_words)

    # Extract keywords
    keywords = rake_object.run(question)


    if isOpen:
      return keywords[len(keywords)-1][0]
    else:
      return keywords[0][0]
```

Figure 5: Function to get the keyword of the sentence

2.1.3 Remove verb in keyword

The verbs are not very important, for us, in the keyword list because they are not presented in the titles of the Wikipedia articles. We have then decided to remove the verbs to have more reliability in the good keyword detection.

To remove the verbs in the keyword list (Figure 6), we have used the package *"nltk"*. This package detects if a word is a verb even if it is conjugated in different tenses. If we detect one, we remove it from the list of keywords or if it is in a string of characters, we just delete the verb from the string.



```python
def removeVerb(keywords):

    text = keywords
    text = nltk.word_tokenize(text)
    result = nltk.pos_tag(text)

    verb = ""

    for array in result:
        if(array[1][:2]) == "VB":
            verb = array[0]

    new_text = keywords


    if verb in new_text:
        querywords = new_text.split()

        resultwords  = [word for word in querywords if word.lower() not in verb]
        new_text = ' '.join(resultwords)


    print("\n4. Keyword without verb : " +new_text)

    return new_text
```

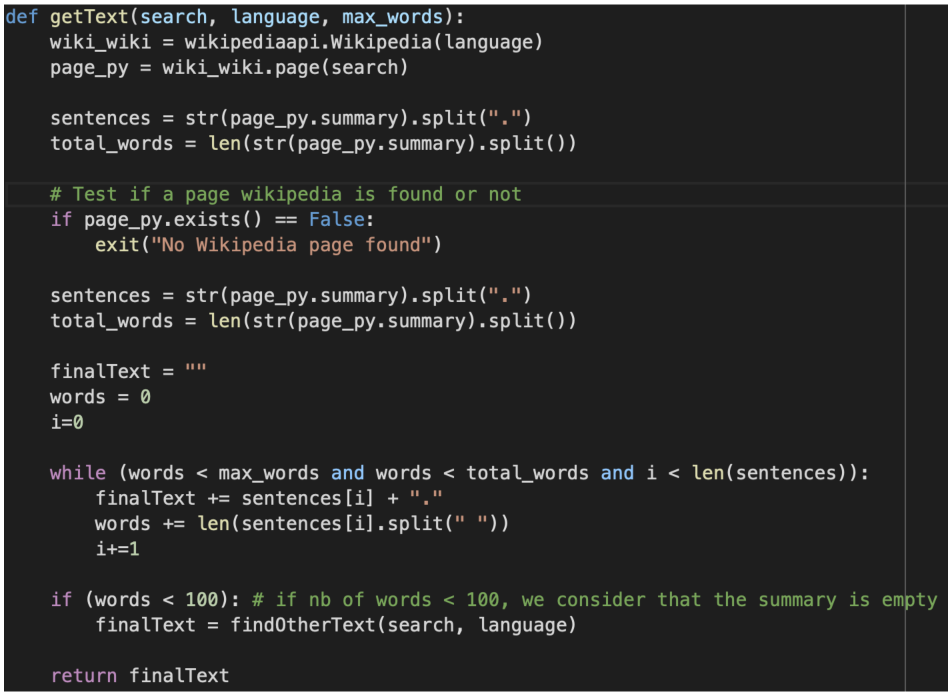
Figure 6: Function to get the remove the verbs in the keywords list

2.2 Get data

2.2.1 Get summary

To get the summary of the keywords (Figure 7), we have used the *"wikiepdia-api"*. To retrieve a good summary to use for the questions, we can pass the number maximum of words we want in it. We check also that the Wikipedia page exists. If we can retrieve information, we split these information by the point of end of sentence : ".". Thanks to that, we have the possibility to count more easily the number of words and to not cut a sentence in two parts.

If we have less than a hundred words, we consider that the summary does not have enough words to use it for the question answering. So, we use another API which is named *"wikipedia"*. This API will help us for some questions, or it will return us a summary with a greater number of words. We will see this package after.



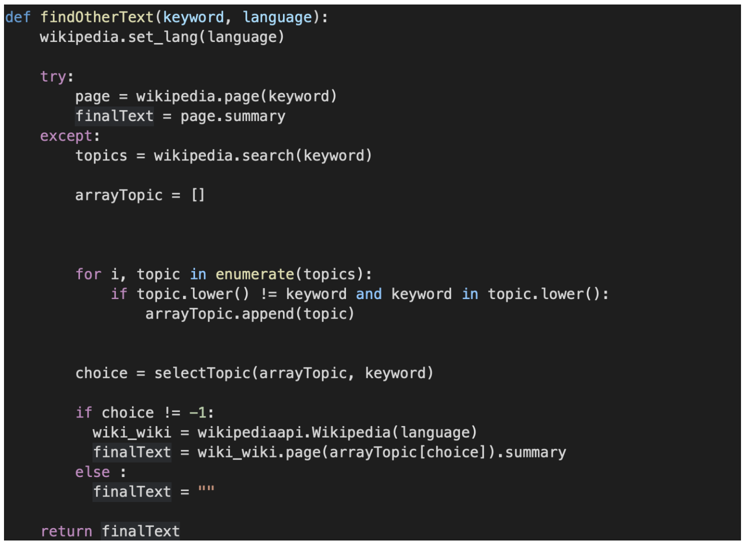
```
def getText(search, language, max_words):
    wiki_wiki = wikipediaapi.Wikipedia(language)
    page_py = wiki_wiki.page(search)

    sentences = str(page_py.summary).split(".")
    total_words = len(str(page_py.summary).split())

    # Test if a page wikipedia is found or not
    if page_py.exists() == False:
        exit("No Wikipedia page found")

    sentences = str(page_py.summary).split(".")
    total_words = len(str(page_py.summary).split())

    finalText = ""
    words = 0
    i=0

    while (words < max_words and words < total_words and i < len(sentences)):
        finalText += sentences[i] + "."
        words += len(sentences[i].split(" "))
        i+=1

    if (words < 100): # if nb of words < 100, we consider that the summary is empty
        finalText = findOtherText(search, language)

    return finalText
```

Figure 7: Function to get the summary according to the keyword

2.2.2 Select topic

For some questions, the *"wikipedia-api"* will not found a summary. For instance : "*Qu'est-ce-qu'un marteau ?*", the page for the word "*marteau*" does not exist in Wikipedia. Our question is about the tool. The Wikipedia page for this question is : "*Marteau (Outil)*".

So, we have used the package *"wikipedia"* which gives us some possible words that can match with our keyword. Thanks to that, the user have the choice between multiple titles that corresponds to his first request. When the user has chosen the keyword that best fits, we get the summary of this choice (we notice that the choices which are proposed to the user, by the API, necessarily exist). Finally, when the user has made a choice and we have retrieved the summary, we return this summary (Figure 8).

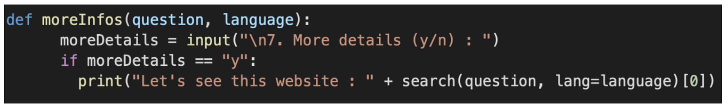


```python
def findOtherText(keyword, language):
    wikipedia.set_lang(language)

    try:
        page = wikipedia.page(keyword)
        finalText = page.summary
    except:
        topics = wikipedia.search(keyword)

        arrayTopic = []



        for i, topic in enumerate(topics):
            if topic.lower() != keyword and keyword in topic.lower():
                arrayTopic.append(topic)


        choice = selectTopic(arrayTopic, keyword)

        if choice != -1:
          wiki_wiki = wikipediaapi.Wikipedia(language)
          finalText = wiki_wiki.page(arrayTopic[choice]).summary
        else :
          finalText = ""

    return finalText
```

Figure 8: Function to get another summary according to the keyword

2.2.3 Get more information

When the user has the response to his question, VEEMbot will propose to him more information about the subject of his question. To do that, we have used the *"googlesearch-python"* API. Finally, if the user wants more information about the subject, VEEMbot will give him a link which is the first link about the question we can find on Google (Figure 9).



```python
def moreInfos(question, language):
      moreDetails = input("\n7. More details (y/n) : ")
      if moreDetails == "y":
        print("Let's see this website : " + search(question, lang=language)[0])
```

Figure 9: Function to get more information about the subject

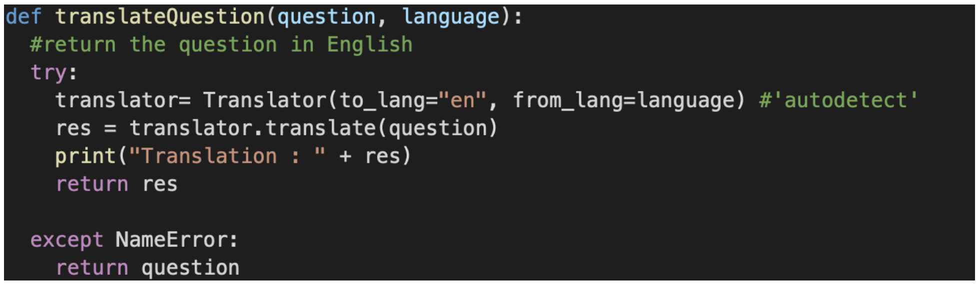
2.3 Find answer

2.3.1 Questions

VEEMbot analyses the request of the user to know whether it is a question or not. To do so, we first check if the request is a calculation or not (See Figure 18). After that, we have two possibilities :

- The question is in English
- The question is not in English

If the question is not in English, we translate it (Figure 10). To that end, we use the *"translator"* package. However, if the question is in English, we do nothing.



```python
def translateQuestion(question, language):
  #return the question in English
  try:
    translator= Translator(to_lang="en", from_lang=language) #'autodetect'
    res = translator.translate(question)
    print("Translation : " + res)
    return res

  except NameError:
    return question
```

Figure 10: Translation of a question in English

When the translation of the question is done (after the last part), we can analyse the question. For this purpose, we have defined all the possible relative pronouns for the question such as *When*, *What*, ... We have also defined all the modal elements we can find in a question like *can*, *could*, *would*, ... So if we found one of this adverbs we have a possibility that the request can be a open-ended question. Then, we detect where is the subject and the verbs (Figure 11 and 23) via the *"spacy"* and *"nltk"* packages. When we have the position of the verb and the subject, we can deduce if the request is a question or not (the verb is after the subject in a question, it depends if there is a modal or not in the question).

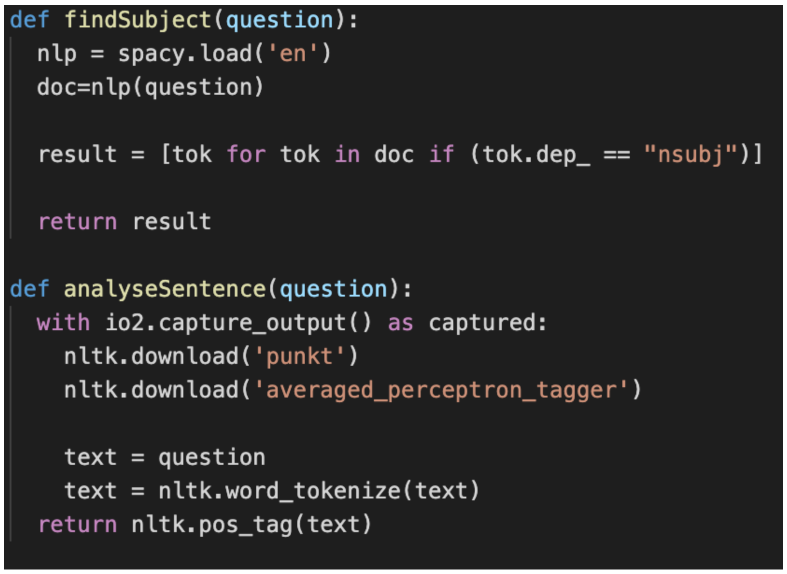

```
def findSubject(question):
  nlp = spacy.load('en')
  doc=nlp(question)

  result = [tok for tok in doc if (tok.dep_ == "nsubj")]

  return result

def analyseSentence(question):
  with io2.capture_output() as captured:
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')

    text = question
    text = nltk.word_tokenize(text)
  return nltk.pos_tag(text)
```

Figure 11: First part of the functions to analyse the question

2.3.1.1 Closed-ended

To answer to YES/NO questions, we have used a model we have found on the Internet and we have trained it with the BoolQ dataset. We have created another version of the BoolQ dataset which does not exist yet : the French version. Indeed, we have now two versions of the BoolQ dataset : one in English and its translation in French.

We have saved the models, for this type of question, to reuse them after without train all the model again. It is a gain in time and resources. We use also the *"torch"* package to load the models linked to this type of question (Figure 12).

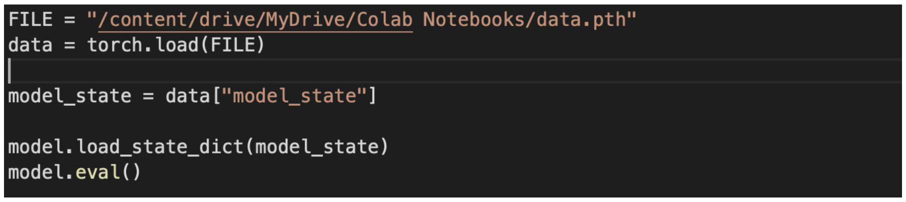
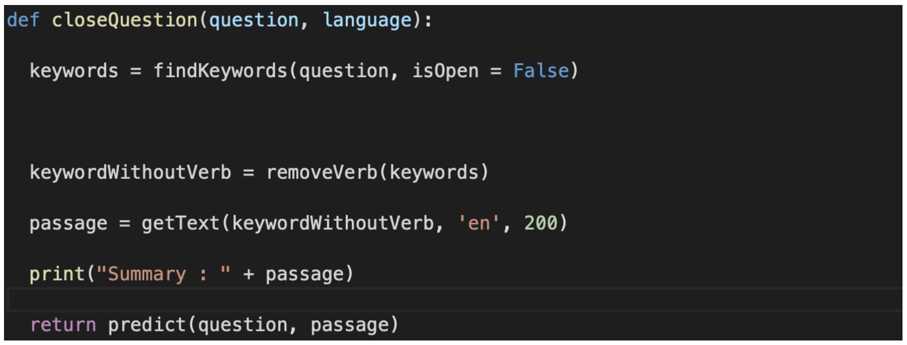

```
FILE = "/content/drive/MyDrive/Colab Notebooks/data.pth"
data = torch.load(FILE)

model_state = data["model_state"]

model.load_state_dict(model_state)
model.eval()
```

Figure 12: Loading the model for the YES/NO questions

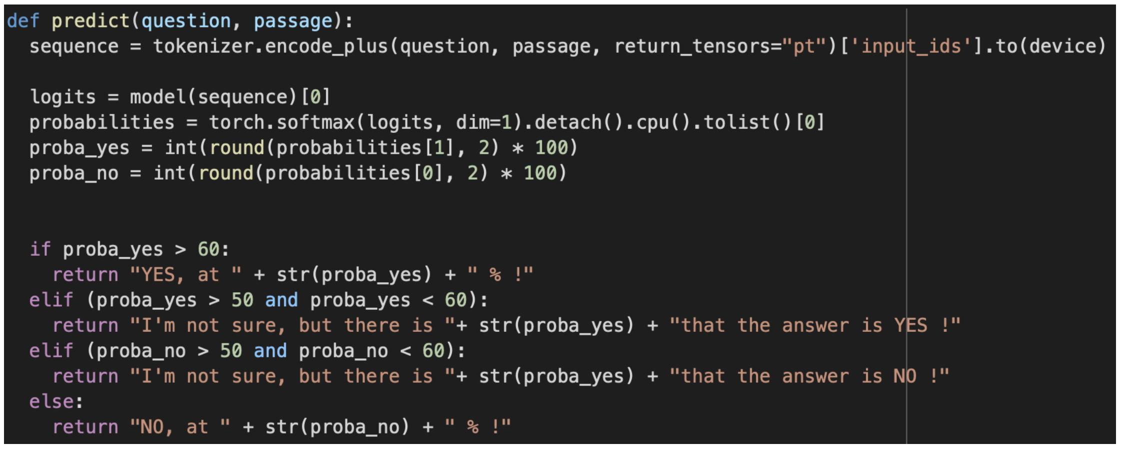
When a user ask a question which is closed-ended, we first retrieve the keyword from the question and we remove the verb. After that, we get the summary according to the keyword we have found before. Finally, we return the predicted result (Figure 13).



```python
def closeQuestion(question, language):

  keywords = findKeywords(question, isOpen = False)


  keywordWithoutVerb = removeVerb(keywords)

  passage = getText(keywordWithoutVerb, 'en', 200)

  print("Summary : " + passage)

  return predict(question, passage)
```

Figure 13: First part of a prediction of YES/NO question

In Figure 14, we encode the question and the summary to then predict the answer. After that, we compute the prediction probabilities to know if the AI is reliable or not. We return a message which is modified according to the probabilities of safety of the response.



```python
def predict(question, passage):
  sequence = tokenizer.encode_plus(question, passage, return_tensors="pt")['input_ids'].to(device)

  logits = model(sequence)[0]
  probabilities = torch.softmax(logits, dim=1).detach().cpu().tolist()[0]
  proba_yes = int(round(probabilities[1], 2) * 100)
  proba_no = int(round(probabilities[0], 2) * 100)


  if proba_yes > 60:
    return "YES, at " + str(proba_yes) + " % !"
  elif (proba_yes > 50 and proba_yes < 60):
    return "I'm not sure, but there is "+ str(proba_yes) + "that the answer is YES !"
  elif (proba_no > 50 and proba_no < 60):
    return "I'm not sure, but there is "+ str(proba_yes) + "that the answer is NO !"
  else:
    return "NO, at " + str(proba_no) + " % !"
```

Figure 14: Second part of a prediction of YES/NO question

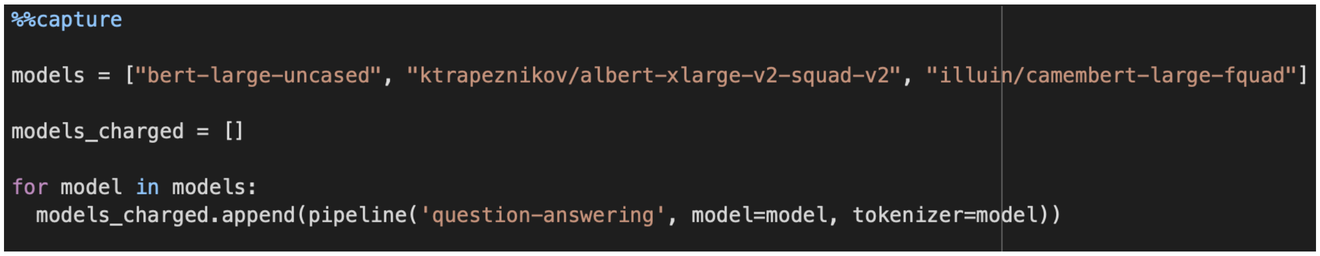
2.3.1.2 Open-ended

First of all, we load into memory all the models we need to answer concerning open-ended questions (Figure 15).

The *"bert-large-uncased"* model is used to respond to open-ended questions in different languages. This cannot be very accurate compared to models trained in a specified language. It will be used for languages that are not French or English.

The second model loaded into memory is *"ktrapeznikov/albert-xlarge-v2-squad-v2"*. This model is trained to respond to questions in the English language. It will be used for all the questions asked in this language.

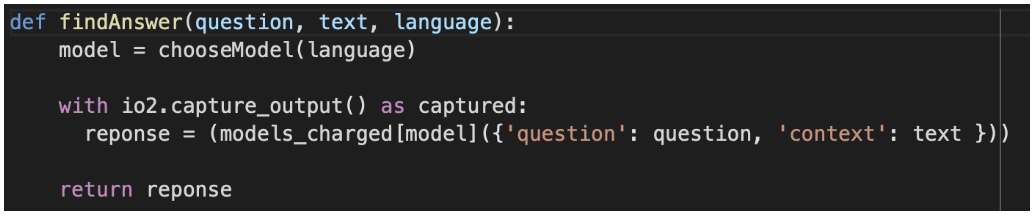
The third model loaded into memory is *"illuin/camembert-large-fquad"*. This model is trained to respond to questions in the French language. It will be used for all the questions asked in this language.



```
%%capture

models = ["bert-large-uncased", "ktrapeznikov/albert-xlarge-v2-squad-v2", "illuin/camembert-large-fquad"]

models_charged = []

for model in models:
  models_charged.append(pipeline('question-answering', model=model, tokenizer=model))
```

Figure 15: Loading of the transformers models

In the Figure 16, the function *"chooseModel"* selects the model according to the language of the user. When we have the good model, we can ask the AI (based on the model we have chosen) to respond to the question via the summary to get data. Finally, we just return the response given by the AI.

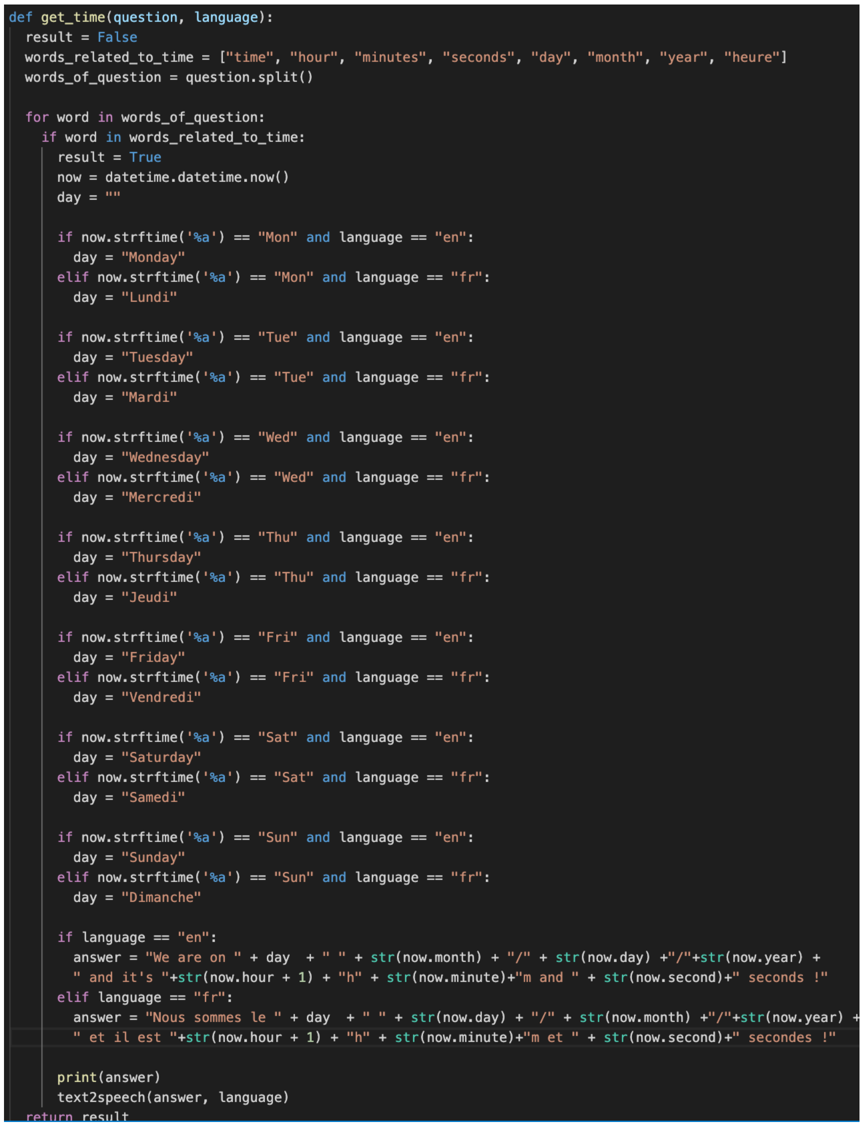


```
def findAnswer(question, text, language):
    model = chooseModel(language)

    with io2.capture_output() as captured:
      reponse = (models_charged[model]({'question': question, 'context': text }))

    return reponse
```

Figure 16: Get a response from the model

2.3.1.3 Time-depending

Here in Figure 17, if we detect a question related to the time, we will give the current time to the user. We are using the *"datetime"* package to determine the current time. After that, we get all the information about the time. According to the language, we create a string with the day of the week, the date, the month, the year, the hour, the minute and the seconds. Then, the AI will tell the user this current date and finally print the string and return if the question is related to time or not.



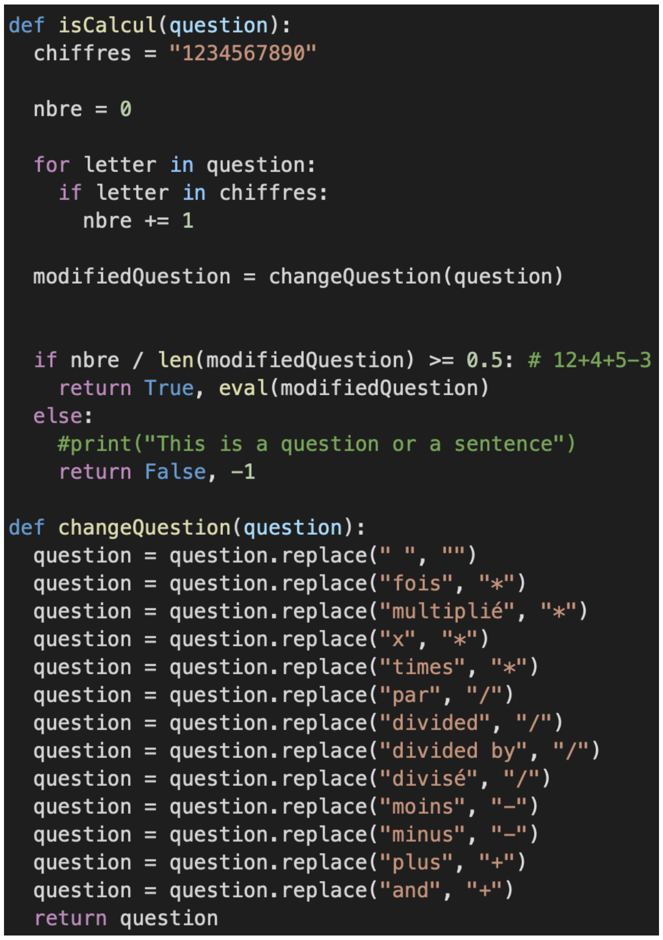
```python
def get_time(question, language):
  result = False
  words_related_to_time = ["time", "hour", "minutes", "seconds", "day", "month", "year", "heure"]
  words_of_question = question.split()

  for word in words_of_question:
    if word in words_related_to_time:
      result = True
      now = datetime.datetime.now()
      day = ""

      if now.strftime('%a') == "Mon" and language == "en":
        day = "Monday"
      elif now.strftime('%a') == "Mon" and language == "fr":
        day = "Lundi"

      if now.strftime('%a') == "Tue" and language == "en":
        day = "Tuesday"
      elif now.strftime('%a') == "Tue" and language == "fr":
        day = "Mardi"

      if now.strftime('%a') == "Wed" and language == "en":
        day = "Wednesday"
      elif now.strftime('%a') == "Wed" and language == "fr":
        day = "Mercredi"

      if now.strftime('%a') == "Thu" and language == "en":
        day = "Thursday"
      elif now.strftime('%a') == "Thu" and language == "fr":
        day = "Jeudi"

      if now.strftime('%a') == "Fri" and language == "en":
        day = "Friday"
      elif now.strftime('%a') == "Fri" and language == "fr":
        day = "Vendredi"

      if now.strftime('%a') == "Sat" and language == "en":
        day = "Saturday"
      elif now.strftime('%a') == "Sat" and language == "fr":
        day = "Samedi"

      if now.strftime('%a') == "Sun" and language == "en":
        day = "Sunday"
      elif now.strftime('%a') == "Sun" and language == "fr":
        day = "Dimanche"

      if language == "en":
        answer = "We are on " + day  + " " + str(now.month) + "/" + str(now.day) +"/"+str(now.year) +
        " and it's "+str(now.hour + 1) + "h" + str(now.minute)+"m and " + str(now.second)+" seconds !"
      elif language == "fr":
        answer = "Nous sommes le " + day  + " " + str(now.day) + "/" + str(now.month) +"/"+str(now.year) +
        " et il est "+str(now.hour + 1) + "h" + str(now.minute)+"m et " + str(now.second)+" secondes !"

      print(answer)
      text2speech(answer, language)
  return result
```

Figure 17: The time depending question

2.3.2 Calculation

When the user ask an open-ended question, we check if this question is a calculation or not (Figure 18). To check if it is a calculation or not, we check if the percentage of numbers in the sentence is more than a certain percentage. If it is not a calculation, we return False (for no calculation) and -1 (for no result available).

For each calculation, strings like *plus*, *minus* or other are replaced by the mathematics signs related to the words. The words supported are in French and English languages only. After that, we compute the result of the calculation and finally, we return True and the result of the calculation we have found. We have used the *"eval"* function to calculate it.



```python
def isCalcul(question):
  chiffres = "1234567890"

  nbre = 0

  for letter in question:
    if letter in chiffres:
      nbre += 1

  modifiedQuestion = changeQuestion(question)


  if nbre / len(modifiedQuestion) >= 0.5: # 12+4+5-3
    return True, eval(modifiedQuestion)
  else:
    #print("This is a question or a sentence")
    return False, -1

def changeQuestion(question):
  question = question.replace(" ", "")
  question = question.replace("fois", "*")
  question = question.replace("multiplié", "*")
  question = question.replace("x", "*")
  question = question.replace("times", "*")
  question = question.replace("par", "/")
  question = question.replace("divided", "/")
  question = question.replace("divided by", "/")
  question = question.replace("divisé", "/")
  question = question.replace("moins", "-")
  question = question.replace("minus", "-")
  question = question.replace("plus", "+")
  question = question.replace("and", "+")
  return question
```

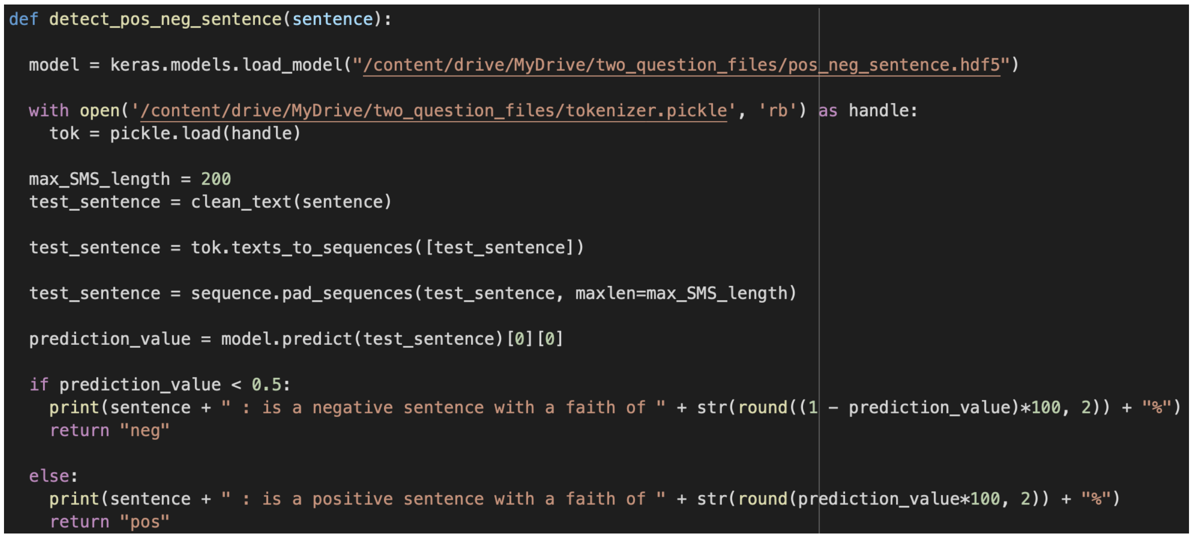
Figure 18: Calculation

2.3.3 Sentence

If we detect that the request of the user is a sentence, VEEMbot will analyze if it is a positive or negative sentence. To do that, we load into memory two elements :

- The model,
- The tokenizer.

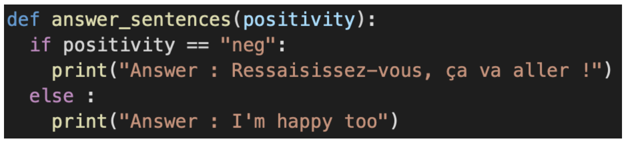
We use the *"keras"* and *"pickle"* to load respectively the model for the AI detection and the "tokenizer" for the sentence. After that, we will clean the sentence. This will replace some words such as *isn't* to *is not* to have a better accuracy in the prediction results. Then, we will transform into sequences the string found and we can finally predict the result concerning the sentence (Figure 19).



```
def detect_pos_neg_sentence(sentence):

  model = keras.models.load_model("/content/drive/MyDrive/two_question_files/pos_neg_sentence.hdf5")

  with open('/content/drive/MyDrive/two_question_files/tokenizer.pickle', 'rb') as handle:
    tok = pickle.load(handle)

  max_SMS_length = 200
  test_sentence = clean_text(sentence)

  test_sentence = tok.texts_to_sequences([test_sentence])

  test_sentence = sequence.pad_sequences(test_sentence, maxlen=max_SMS_length)

  prediction_value = model.predict(test_sentence)[0][0]

  if prediction_value < 0.5:
    print(sentence + " : is a negative sentence with a faith of " + str(round((1 - prediction_value)*100, 2)) + "%")
    return "neg"

  else:
    print(sentence + " : is a positive sentence with a faith of " + str(round(prediction_value*100, 2)) + "%")
    return "pos"
```

Figure 19: Function to detect if the sentence is positive or negative

When we have detected if the sentence is positive or negative, we will just return an answer like *"Pull yourself together, you'll be fine!"* for the negative sentence and *"I'm happy too!"* for the positive ones (20). Even though it could have been interesting to develop this part, it was not our real goal of our project, so we did not go further in the development of this part. We focused on the main part.



```
def answer_sentences(positivity):
  if positivity == "neg":
    print("Answer : Ressaisissez-vous, ça va aller !")
  else :
    print("Answer : I'm happy too")
```

Figure 20: Function showing the response according to the sentiment of the sentence

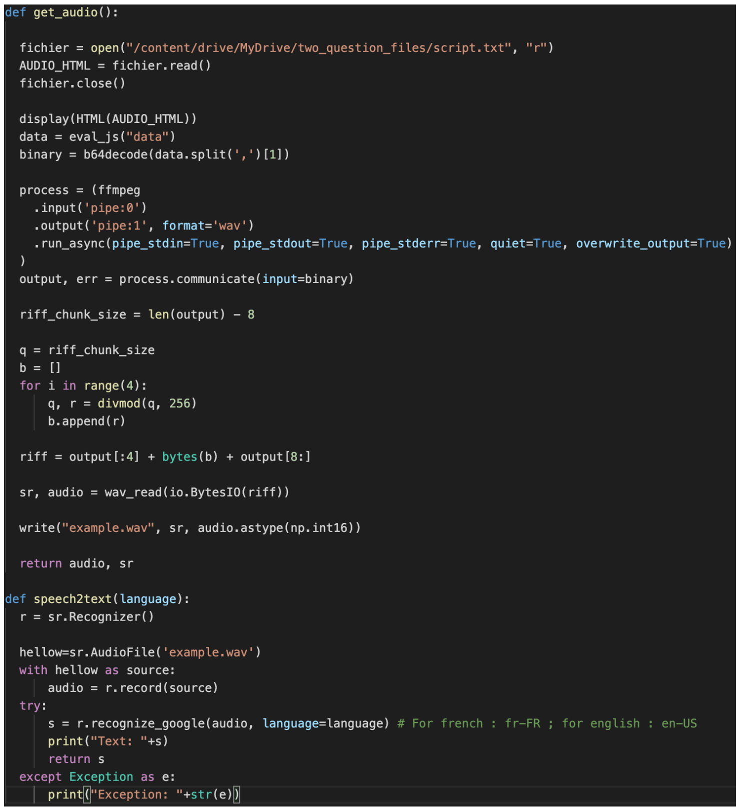
2.4 Communicate orally

2.4.1 Speech to text

The user has the possibility to talk to the assistant to avoid the usage of the keyboard. The packages *"gtts"*, *"ffmped-python"* and *"SpeechRecognition"* will help us to do that. As we are working on Google Colab, we have encountered some problems to save the voice of the user because when we use the *"pyaudio"* package, no microphone is detected. So, we have found one code which works to save the voice of the user when we are using Google Colab but it is a little bit more difficult than the previous that uses *"pyaudio"* (Figure 21).

The *"get_audio"* function will help us to display a button to stop the record of the audio. When we click on this button, the audio will be saved into the audio file *"example.wav"*.

The second function *"speech2text"* will recognize all the words spoken by the user and transform the audio to a text. This text will be returned to continue the treatment of the request of the user.

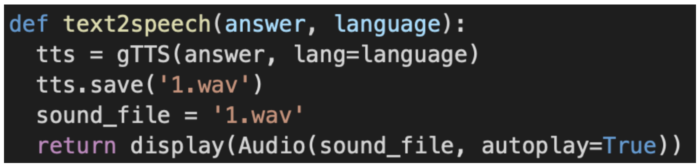


```
def get_audio():

  fichier = open("/content/drive/MyDrive/two_question_files/script.txt", "r")
  AUDIO_HTML = fichier.read()
  fichier.close()

  display(HTML(AUDIO_HTML))
  data = eval_js("data")
  binary = b64decode(data.split(',')[1])

  process = (ffmpeg
    .input('pipe:0')
    .output('pipe:1', format='wav')
    .run_async(pipe_stdin=True, pipe_stdout=True, pipe_stderr=True, quiet=True, overwrite_output=True)
  )
  output, err = process.communicate(input=binary)

  riff_chunk_size = len(output) - 8

  q = riff_chunk_size
  b = []
  for i in range(4):
      q, r = divmod(q, 256)
      b.append(r)

  riff = output[:4] + bytes(b) + output[8:]

  sr, audio = wav_read(io.BytesIO(riff))

  write("example.wav", sr, audio.astype(np.int16))

  return audio, sr

def speech2text(language):
  r = sr.Recognizer()

  hellow=sr.AudioFile('example.wav')
  with hellow as source:
      audio = r.record(source)
  try:
      s = r.recognize_google(audio, language=language) # For french : fr-FR ; for english : en-US
      print("Text: "+s)
      return s
  except Exception as e:
      print("Exception: "+str(e))
```

Figure 21: Functions to save the audio of the user

2.4.2 Text to speech

When the user will ask his question orally, VEEMbot will respond orally too for any type of sentences. To make it possible, we use the *"gtts"* package to transform text to audio. This audio will be saved as *"1.wav"*. Finally, we use the function *"Audio"* of the *"IPython"* library to launch the audio of this file. However, we can encountered problems. This is the reason why we have used the *"display"* function of the *"IPython"* package too (Figure 22).



```
def text2speech(answer, language):
  tts = gTTS(answer, lang=language)
  tts.save('1.wav')
  sound_file = '1.wav'
  return display(Audio(sound_file, autoplay=True))
```

Figure 22: Function to make VEEMbot talking

3 Remaining work

3.1 Problems

There is still some problems remaining in VEEMbot :

- The detection of the keyword
- The question type detection
- The transformation of a speech to a text
- The correction of the question

The detection of the keyword could work better. Indeed, most of the times, the detection of the keyword is good but sometimes the detection function does not detect the good keyword.

The question type detection is simple. We don't use an AI or some complex tricks to detect if the sentence is a question and if it is a question, what type of question, open-ended or closed-ended question, it is.

The transformation of a speech to a text is not very accurate due to the accent of the user who is talking or due to the noise behind the voice of the user for instance. But most of the time, the algorithm retrieves the good sentence spoken by the user.

We have implemented the correction of the errors in the question into VEEMbot but as we have encountered some problems, we kept the code into VEMMbot but it is not used. Indeed, sometimes, the correction of the errors does not correct some errors but can also add some into the question. For example, a question like *"Where is Montbéliard ?"* can be transformed into *"Where is Monthéliard ?"*. *"Montbéliard"* was well written in the question (with a *"b"*) but it has been corrected into something wrong (with a *"h"*). So, finally, after multiple modifications of this part, we have decided to keep it but not use it due to some real problems of accuracy.

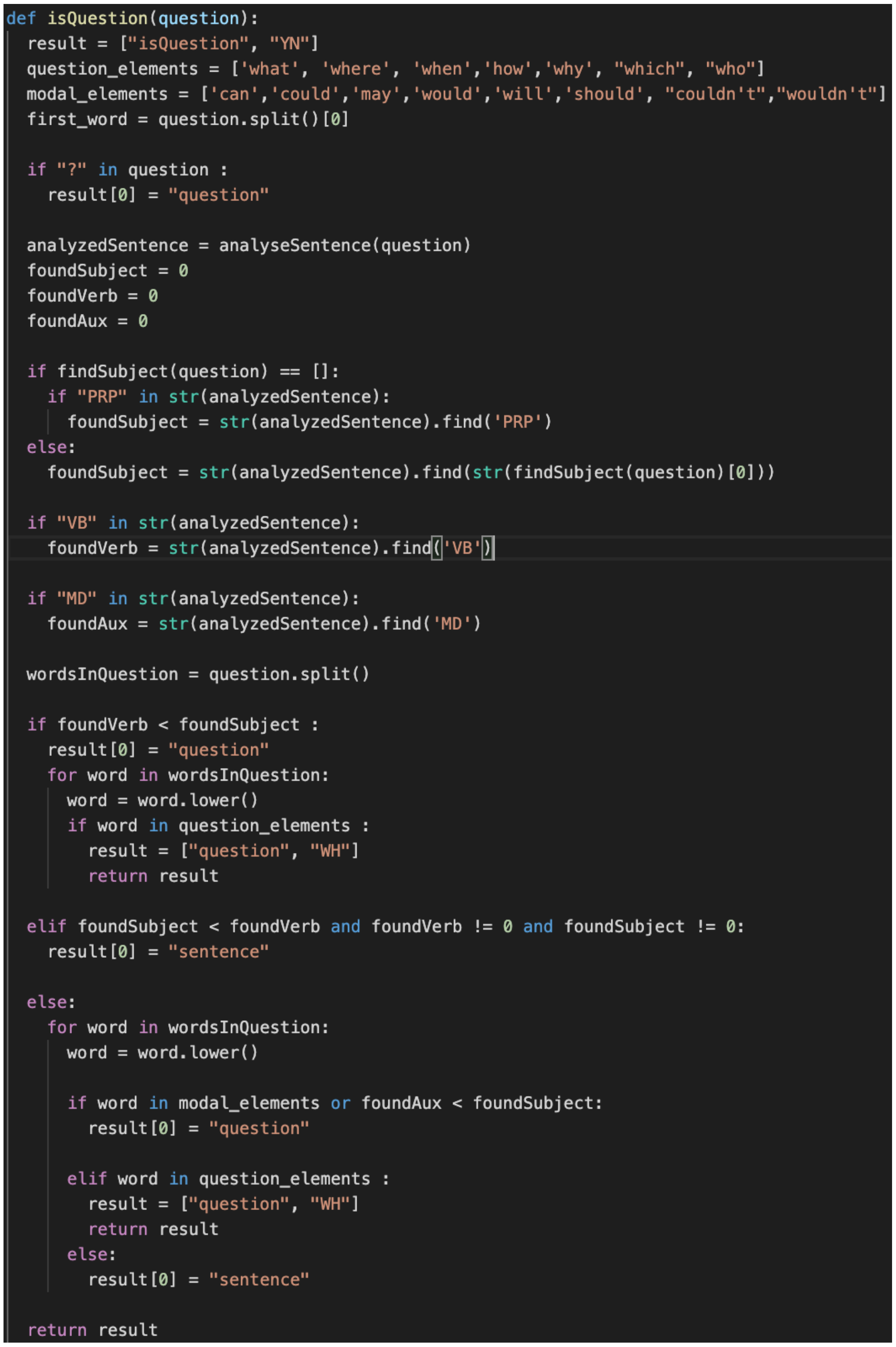
3.2 Possible improvements

Of course, some of the improvements are directly linked to the problems we have described before. The detection of the keyword can be improved by using an AI or by doing more data processing on the request of the user. The question type detection can be improved by using an AI for example. It exists some AI that we have tested on the Internet, but no one of these AIs satisfied us in the results. So we have decided to make our solution. As for the transformation of a speech to a text, it exists some functions to remove the noise behind the user's voice. Finally, the correction of errors in a question could be done by an AI or to modify the results of the different functions we have tested.

Another improvement which could be done is using a little server on which we just need to make requests and get the response of it. By doing this, we can develop the front-end of our application in multiple languages like JavaScript for the web part (which can be added to website) or Python for a classic application to exchange with VEEMbot. With this solution, all of the large computer processing will be done by the server and not by the computer of the user. By doing so, we could implement it a small computer (Arduino, Raspberry Pi, ...) because it would save battery life, and memory.

The last improvement is about the possible lines of codes which can be redundant in the execution. Effectively, we have tried to delete all the execution which are duplicates but some of these calls can still be there.

List of figures



```python
def isQuestion(question):
  result = ["isQuestion", "YN"]
  question_elements = ['what', 'where', 'when','how','why', "which", "who"]
  modal_elements = ['can','could','may','would','will','should', "couldn't","wouldn't"]
  first_word = question.split()[0]

  if "?" in question :
    result[0] = "question"

  analyzedSentence = analyseSentence(question)
  foundSubject = 0
  foundVerb = 0
  foundAux = 0

  if findSubject(question) == []:
    if "PRP" in str(analyzedSentence):
      foundSubject = str(analyzedSentence).find('PRP')
  else:
    foundSubject = str(analyzedSentence).find(str(findSubject(question)[0]))

  if "VB" in str(analyzedSentence):
    foundVerb = str(analyzedSentence).find('VB')

  if "MD" in str(analyzedSentence):
    foundAux = str(analyzedSentence).find('MD')

  wordsInQuestion = question.split()

  if foundVerb < foundSubject :
    result[0] = "question"
    for word in wordsInQuestion:
      word = word.lower()
      if word in question_elements :
        result = ["question", "WH"]
        return result

  elif foundSubject < foundVerb and foundVerb != 0 and foundSubject != 0:
    result[0] = "sentence"

  else:
    for word in wordsInQuestion:
      word = word.lower()

      if word in modal_elements or foundAux < foundSubject:
        result[0] = "question"

      elif word in question_elements :
        result = ["question", "WH"]
        return result
      else:
        result[0] = "sentence"

  return result
```

Figure 23: Second part of the functions to analyse the question