TP 5 : ASSISTANT VOCAL

```
NOM – PRENOM :
Déposer le notebook sur moodle en fin de séance et une archive avec
vos grammaires et fichiers de test et fichiers résultats
```

Environnement de travail

Travailler sous la machine virtuelle dédiée aux TP de PAROLE et activer l'environnement conda dédié au TP_TAP_SRI2

```
conda info –e : pour connaitre les environnement disponibles s
i besoin
```

Exécuter les deux cellules suivantes après avoir modifié le chemin d'accès à la grammaire

Entrée [ ]:
```
import nltk
from nltk import *
from nltk import load_parser
import os
from os import *
import string

# PATH = 'chemin vers votre dossier de TP'
PATH = '/Users/ferrane/Documents/ENSEIGNEMENT/SRI-2A/COURS_TAP/COURS_T
chdir(PATH + 'TP5_ASSISTANT_VOCAL/TP5_ASSISTANT_VOCAL_CORRECTION')
```

Entrée [ ]:
```
!pwd; ls
```

PARTIE 1 : Grammaire, Analyseur, Chaine à traiter et Résultats

Question 1 : prise en main du fonctionnement d'un analyseur

Exécuter chacune des cellules suivantes et répondre aux questions dans des zones de texte dédiées

Entrée [ ]:
```
# CELLULE 1
# Chargement de la grammaire
nom_fichier_grammaire_V1 = 'GRAMMAIRE_ASSISTANT_V1.fcfg'
# Visualisation du contenu de la grammaire
data.show_cfg(nom_fichier_grammaire_V1)
```

1.1) Quel langage décrit la grammaire contenue dans le fichier GRAMMAIRE_ASSISTANT_V1.fcfg

Entrée [ ]:
```
# CELLULE 2
# Création d'un analyseur basé sur la grammaire chargée
# version avec un niveau de trace détaillé

analyseur_V1 = parse.load_parser(nom_fichier_grammaire_V1, trace=2)

# ATTENTION MEME SI ON RELANCE LA CELLULE, LES MODIFS FAITES SUR LA GF
# IL FAUT RELANCER LE NOYAU en attendant mieux.

# POUR AFFICHER SANS LE NIVEAU DE TRACE DETAILLE
# analyseur_notrace = parse.load_parser(nom_fichier_grammaire)
```

1.2) Quel est le rôle de l'analyseur créé ?

Entrée [ ]:
```
# CELLULE 3 = Fonction de test d'un ensemble d'exemples

def tester_par_lot(Dico_test, analyseur):
    resultat_analyse_lot = {}

    for k in Dico_test.keys():
        print(k)
        tokens = Dico_test[k].split()
        print(tokens)
        #Transformation de la chaine à traiter en liste de tokens
        #resultat_analyse_lot[k] = analyseur_V1.parse(tokens)
        resultat_analyse_lot[k] = analyseur.parse(tokens)
        #print(resultat_analyse_lot[k])
        print("=========================================\n")

    return resultat_analyse_lot
```

1.3) Quelles sont les entrées-sortie de la fonction tester_par_lot ?

Entrée [ ]:
```
# CELLULE 4

exemple_test_G1 = { 'exemple1' : 'cinq plus trois', 'exemple2' : 'six

resultat_test_G1 = tester_par_lot(exemple_test_G1, analyseur_V1)
```

1.4) Commenter de manière détaillée un des deux résultats obtenus

QUESTION 2 : Evolution de la grammaire - flexibilité du langage

a) Faire une copie de la grammaire dans le fichier GRAMMAIRE_ASSISTANT_V2.fcfg

b) modifier cette version de façon à prendre en compte :
- des chaines comme '5 + 3' ou '5 moins sept'
- d'autres nombres 'trente', 'quinze', ...
- et d'autres opérateurs binaires : divisé par (/), multiplié par (*), fois (*), puissance (^) ...

--> Pour simplifier on considèrera les opérateurs composés comme un seul mot en remplaçant ' ' par '_'
'multiplié_par', 'divisé_par'

2.1) Exécuter les trois cellules suivantes pour tester cette version de la grammaire et le fonctionnement de
l'analyseur correspondant.

Entrée [ ]:
```
# CELLULE 1 = CHARGEMENT GRAMMAIRE G2

# Chargement de la deuxième version de la grammaire
nom_fichier_grammaire_V2 = 'GRAMMAIRE_ASSISTANT_V2.fcfg'
# Visualisation du contenu de la grammaire
data.show_cfg(nom_fichier_grammaire_V2)
```

Entrée [ ]:
```
# CELLULE 2 = Création d'un analyseur basé sur la grammaire G2
# version avec un niveau de trace détaillé

analyseur_V2 = parse.load_parser(nom_fichier_grammaire_V2, trace=2)

# ATTENTION MEME SI ON RELANCE LA CELLULE, LES MODIFS FAITES SUR LA GR
# IL FAUT RELANCER LE NOYAU en attendant mieux.

# version sans niveau de trace
#analyseur_notrace = parse.load_parser(nom_fichier_grammaire)
```

Entrée [ ]:
```
# CELLULE 3 = Test sur des exemples spécifiques

exemple_test_G2 = { 'exemple1' : '5 + 3', 'exemple2' : '6 - 2', 'exemp
                    'exemple4' : 'cinq fois trois', 'exemple5' : 'neuf
                    'exemple6': 'vingt puissance trois'}

resultat_test_G2 = tester_par_lot(exemple_test_G2, analyseur_V2)
```

2.2) Verifier les résultats correspondant à chacune des phrase. Quel résultat obtient-on sans le mode trace ?

Entrée [ ]:
```
analyseur_V2_no_trace = parse.load_parser(nom_fichier_grammaire_V2)
resultat_test_G2 = tester_par_lot(exemple_test_G2, analyseur_V2_no_tra
```

QUESTION 3 : Aspect multi-tâche

a) Faire une copie de la grammaire G2 dans le fichier GRAMMAIR
E_ASSISTANT_V3.fcfg

b) modifier cette version de façon à prendre aussi en compte u
ne nouvelle tâche comme :
- 'appelle le 0 5 6 7 8'
- 'envoie un sms au 0 5 6 7 8'
- 'appelle Paul'
- 'envoie un texto à Polo'

--> Pour simplifier on considèrera les commandes composée
s comme un seul mot en remplaçant ' ' par '_'
'appelle_le', 'envoie_un_sms_à', ...

3.1) Exécuter les cellules suivantes pour tester cette version de la
grammaire et le fonctionnement de
l'analyseur correspondant.

Entrée [ ]:
```
# CELLULE 1 = Chargement de la grammaire G3

nom_fichier_grammaire_V3 = 'GRAMMAIRE_ASSISTANT_V3.fcfg'
# Visualisation du contenu de la grammaire
data.show_cfg(nom_fichier_grammaire_V3)
```

Entrée [ ]:
```
# CELLULE 2 = Création d'un analyseur basé sur la grammaire G3

#analyseur_V3 = parse.load_parser(nom_fichier_grammaire_V3, trace=2)

# ATTENTION MEME SI ON RELANCE LA CELLULE, LES MODIFS FAITES SUR LA GR
# IL FAUT RELANCER LE NOYAU en attendant mieux.

# version sans niveau de trace
analyseur_V3 = parse.load_parser(nom_fichier_grammaire_V3)
```

Entrée [ ]:
```
def extraire_interpretation(arbre):
    tmp= str(arbre[0])
    racine = tmp.split('\n')
    interpretation = racine[0]+')'
    # print(type(interpretation))
    print('interpretation = ', interpretation)

    indice = interpretation.find('=')
    # print('indice = ', indice)

    chaine_a_interpreter = interpretation[indice+2:len(interpretation)
    print('chaine_a_interpreter = ', chaine_a_interpreter)

    return chaine_a_interpreter
```

3.2) Quelles sont les entrées-sorties de la fonction extraire_interprétation ainsi que son rôle ?

Entrée [ ]:
```
def calculer(operation):
    resultat = eval(operation)
    print('resultat = ', operation, ' = ', resultat )
    return resultat
```

3.3) Quelles sont les entrées-sorties de la fonction calculer ainsi que son rôle ? Tester au besoin

Entrée [ ]:
```
# CELLULE 3 = Test sur exemples spécifiques

exemple_test_G3 = { 'exemple1' : '5 + 3', 'exemple2' : 'six moins deux
                    'exemple4': '1 2 3 4 5', 'exemple5' : 'envoie_un_s
                    'exemple6': 'appelle_le 1 2 3 4 5', 'exemple7':'vi
                    'exemple8' : 'appelle_le cinq fois trois', 'exempl
                    'exemple10':'appelle Polo', 'exemple11': 'vingt pu

resultat_test_G3 = tester_par_lot(exemple_test_G3, analyseur_V3)
# print(resultat_test_G3)

for k in resultat_test_G3.keys():
    for arbre in resultat_test_G3[k] :
        #print(arbre)
        print("--------------------")

        chaine_a_interpreter = extraire_interpretation(arbre)
        print(chaine_a_interpreter)

        tache = chaine_a_interpreter.split(", ")
        commande = tache[0]
        info = ' '.join(tache[1:])

        if commande == 'compute':
            resultat = calculer(info)
            print("resultat = ", resultat )
        else:
            if commande == 'callNumber':
                print('Nous appelons le numéro : ', info)
            else:
                if commande == 'callPerson':
                    print('Nous appelons ', info)
                else:
                    if commande == 'sendTexto':
                        print('vous voulez envoyer un texto au ', info
                    else:
                        print('je n\'ai pas compris votre requête')


        print("..................................\n\n\n")

        print("==========================================================
```

3.4) Commenter de manière détaillée les résultats obtenus

PARTIE 4 : Reconnaissance de la parole et normalisation des transcriptions

S'assurer que les librairies speechrecognition, gtts et pyaudio ont été installées dans l'environnement conda de TP. Les installer au besoin dans ce même environnement (TP_TAP_SRI2)

```
cond list pour avoir la liste de toutes les dépendances instal
lées dans l'environnement.

https://pypi.org/project/SpeechRecognition/  : pip install Spe
echRecognition
```

4.1) Utiliser un outil de reconnaissance exitant (outils "sur étagère"). Exécuter la cellule suivante plusieurs fois et faire un compte rendu des différents tests effectués dans une zone de texte dédiée.

Entrée [ ]:

```
import speech_recognition as sr
import pyaudio
from gtts import gTTS

r = sr.Recognizer()

micro = sr.Microphone()

with micro as source:
    print("Speak!")
    audio_data = r.listen(source)
    print("End!")
result = r.recognize_google(audio_data, language="fr-FR")

# pour une reconnaissance de la parole en anglais
# result = r.recognize_google(audio_data, language="en-EN")

print ("Vous avez dit : ", result)
```

4.2) Compte-rendu de vos tests

```
===

Exemple de commentaires

Enoncé = appele le zéro trois quatre cinq six
Speak!
End!
Vous avez dit :  appelle le 03456
--> résultat correct ou erreur constatée : omission / substitution /
insertion préciser
    ici 7 mots bien reconnus = 0 erreur

===
```

Entrée [ ]:
```
def normaliser_transcription(transcription):

    if "appelle le" in transcription:
        result = transcription.replace("appelle le", "appelle_le")
    elif "envoie un sms à" in transcription:
        result = transcription.replace("envoie un sms à", "envoie_un_s
    else :
        # pas de modification apportée
        result = transcription

    # transformation de la chaine en tokens
    chaine_tokenisee = result.split()
    # print(chaine_tokenisee)

    # cas du numéro de téléphone supposé en fin de chaine
    last = len(chaine_tokenisee)
    numero = chaine_tokenisee[last-1]
    print(numero, type(numero))

    if numero.isdigit() :
        chaine_tokenisee.pop()
        # print('après pop', chaine_tokenisee)
        phone =''
        for c in numero :
            phone = phone + c + ' '
            chaine_tokenisee.append(c)
        # print('phone', phone)
        # print('chaine', chaine_tokenisee)

    return chaine_tokenisee
```

4.3) Quelles sont les entrées-sorties de la fonction normaliser_transcription ?

4.4) Tester cette fonction, proposer d'autres exemples de test. Apporter les modifications nécéssaires selon les besoins de l'assistant vocal (cf. Partie 5). Reporter les résultats de vos tests dans une zone de texte dédiée.

Entrée [ ]:
```
print(normaliser_transcription('appelle le 03456'))
print(normaliser_transcription('envoie un sms à Paul'))
```

PARTIE 5 : Assitant vocal complet = Reconnaissance +

Interprétation + Synthèse

a) Etudier les fonctions mises à votre disposition pur bien co
mprendre leur rôle

Entrée [ ]:
```
nalyse de la phrase à traiter (supposer être le résultat du processus d
f interpreter_commande(commande, analyseur) :

 #Transformation de la chaine à traiter en liste de tokens - A décomme
 #tokens = commande.split()
 #print(tokens)

 resultat_analyse = analyseur.parse(commande)
 print(resultat_analyse)

 for arbre in resultat_analyse :
     chaine_a_interpreter = extraire_interpretation(arbre)
     tache = chaine_a_interpreter.split(", ")
     commande = tache[0]
     info = ' '.join(tache[1:])
     if commande == 'compute':
         resultat = calculer(info)
         message = 'le résultat de votre opération est égal à ' + str(
     else:
         if commande == 'callNumber':
             message = 'OK, j\'appelle le : ' + str(info)
         else:
             if commande == 'sendTexto':
                 message= 'Vous voulez envoyer un message au '+ str(in
             else:
                 if commande =='callPerson':
                     message= 'Nous appelons '+ str(info)
                 else: message = 'Je n\'ai pas compris votre requête'

 return message
```

5.1) Quelles sont les entrées-sorties de la fonction interpreter_commande ainsi que son rôle ?

Entrée [ ]:
```
def text_to_speech(text):
    # Initialize gTTS with the text to convert
    speech = gTTS(text, lang="fr", slow=False)

    # Save the audio file to a temporary file
    speech_file = 'speech.mp3'
    speech.save(speech_file)

    # Play the audio file
    os.system('afplay ' + speech_file)
```

5.2) Quelles sont les entrées-sorties de la fonction text_to_speech ainsi que son rôle ? Tester au besoin

5.3) Tester de manière exhaustive l'assistant vocal ci-après

Entrée [ ]:
```
## CELLULE ASSISTANT VOCAL

import speech_recognition as sr
import pyaudio
from gtts import gTTS

r = sr.Recognizer()

micro = sr.Microphone()

with micro as source:
    print("Speak!")
    audio_data = r.listen(source)
    print("End!")
transcription = r.recognize_google(audio_data, language="fr-FR")

# pour une reconnaissance de la parole en anglais
#result = r.recognize_google(audio_data, language="en-EN")

print ("Vous avez dit : ", transcription)

transcription_normalisee = normaliser_transcription(transcription)
print(transcription_normalisee)

message_reponse = interpreter_commande(transcription_normalisee, analy

#repondre_vocal(transcription_operation, resultat )
text_to_speech(message_reponse)
```

```
5.4) Faire un rapport des résultats

Exemple de résultat
===
Enoncé : appele Paul
Speak!
End!

Vous avez dit :  appelle Paul
['appelle', 'Paul']
<generator object FeatureChart.parses at 0x7f85dc1dd270>
interpretation =  (CONTACTER[INTERP=(callPerson, Paul)])
chaine_a_interpreter =  callPerson, Paul
===
```

RENDU :

DEPOSER LE NOTEBOOK A VOTRE NOM SUR MOODLE (.ipynb et version pdf)
AINSI QUE, SUIVANT LE CAS, LES FICHIERS AUDIO CORRESPONDANT AUX TESTS
DE L'ASSISTANT EFFECTUÉS (dans une archive).

Entrée [ ]: