

```

1  #!/usr/bin/env python
2  # coding: utf-8
3  #*****
4  #Auteure: Eva Rolin
5  #Date: Novembre 2019
6  #*****
7  # Nous importons les modules suivants frenetic https://github.com/hardik-
vala/FreNetic/blob/master/README.md
8  # et spacy
9  #*****
10 from frenetic import *
11 fwn = FreNetic("C:/Users/evaro/OneDrive/Documents/Github/LEXenstein/EVAstein/wolf-1.0b4.xml")
12 import re
13 import spacy
14 nlp = spacy.load("fr_core_news_md")
15 #*****
16 # Cette fonction retourne la longueur d'un candidat
17 #*****
18 def prescripteur_longueur(candidat):
19     compteur_lettre = 0
20     for lettre in candidat:
21         compteur_lettre += 1
22     return compteur_lettre
23 #*****
24 # Cette fonction retourne le nombre de sens trouvés pour un candidat
25 # à l'aide de l'api Frenetic
26 #*****
27 def prescripteur_sens(candidat):
28     liste_definition_bis = ["rien"]
29     try:
30         synsets = fwn.synsets(candidat)
31         element = re.compile("\s+eng-[0-9]+-[0-9]+-[a-z]")
32         resultat = element.findall(str(synsets))
33         result = []
34         for i in resultat:
35             result.append(i.strip())
36         nb = len(result)
37         liste_definition = []
38         compteur = 0
39         if nb > 1:
40             while nb != 0:
41                 for r in result:
42                     syn = fwn.synset(r)
43                     definition = syn.defn()
44                     liste_definition.append(definition)
45                     print(liste_definition)
46                     nb = nb-1
47                     compteur += 1
48             return compteur
49         if nb ==1:
50             for r in result:
51                 syn = fwn.synset(r)
52                 definition = syn.defn()
53                 liste_definition.append(definition)
54                 compteur += 1
55             return compteur
56     except:
57         return 0
58 #*****
59 # Cette fonction permet de retourner la fréquence pour un mot à partir du fichier téléchargé
60 # à l'adresse suivante: http://www.lexique.org/
61 #*****
62 def prescripteur_frequence(candidat):
63     with open("C:/Users/evaro/OneDrive/Documents/Github/LEXenstein/EVAstein/Lexique383.tsv",
encoding="utf-8") as file:
64         next(file)

```

```

65     texte = file.readline()
66     while texte:
67         texte_cut = texte.strip().split("\t")
68         mot = texte_cut[0]
69         candidat = candidat.replace(candidat[0],candidat[0].lower())
70         freq = texte_cut[8]
71         if mot == candidat:
72             return freq
73         texte= file.readline()
74 *****
75 # Cette fonction permet de convertir une liste en une chaine de caractères
76 # afin d'imprimer les candidats dans un format adéquat
77 *****
78 def conversion_string(liste):
79     st = ""
80     try:
81         for elt in liste:
82             st += elt+" "
83         return st
84     except:
85         return st
86
87 *****
88 # Cette fonction prend en entrée un fichier au format victor et donne en sortie un fichier
89 # au format victor avec les candidats ordonnés
90 *****
91 def main_frequence(victor_corpus,vic):
92     with open(victor_corpus,"r") as file:
93         with open(vic,"w") as file2:
94             texte = file.readline()
95             dictionnaire = {}
96             while texte:
97                 texte_cut = texte.strip().split("\t")
98                 phrase = texte_cut[0]
99                 target = texte_cut[1]
100                 index = int(texte_cut[2])
101                 try:
102                     candidat_cut = texte_cut[3]
103                 except:
104                     candidat_cut = ""
105                 cdt = candidat_cut.split()
106                 for candidat in cdt:
107                     freq = prescripteur_frequence(candidat)
108                     if freq is None:
109                         freq == 0
110                     else:
111                         freq = prescripteur_frequence(candidat)
112                     if target not in dictionnaire:
113                         dictionnaire[target] = []
114                         dictionnaire[target].append((candidat,freq))
115                 liste = test_dico_2(dictionnaire,target)
116                 result = conversion_string(liste)
117                 file2.write("{}\t{}\t{}\t{}".format(phrase,target,index,result))
118                 file2.write("\n")
119                 texte = file.readline()
120
121
122 *****
123 # Cette fonction permet de lemmatiser les candidats
124 *****
125 def lemmatize_fr_candidat(candidat,phrase,ident,target):
126     liste = []
127     phrase = phrase.replace(target,candidat)
128     doc2 = nlp(phrase)
129     for element in doc2:
130         liste.append(element.lemma_)
131     return liste[ident]

```

```

132 #*****
133 # Cette fonction prend en argument un fichier victor et retourne un fichier victor
134 # avec les candidats classés du plus facile au plus compliqué
135 #*****
136 def main_sens(victor_corpus,vic):
137     with open(victor_corpus,"r") as file:
138         with open(vic,"w") as file2:
139             texte = file.readline()
140             dictionnaire = {}
141             while texte:
142                 texte_cut = texte.strip().split("\t")
143                 phrase = texte_cut[0]
144                 target = texte_cut[1]
145                 index = int(texte_cut[2])
146                 try:
147                     candidat_cut = texte_cut[3]
148                 except:
149                     candidat_cut = ""
150                 cdt = candidat_cut.split()
151                 for candidat in cdt:
152                     cd = lemmatize_fr_candidat(candidat,phrase,index,target)
153                     nb_sens = prescripteur_sens(cd)
154                     if target not in dictionnaire:
155                         dictionnaire[target] = []
156                     dictionnaire[target].append((candidat,nb_sens))
157                 liste = test_dico_2(dictionnaire,target)
158                 result = conversion_string(liste)
159                 file2.write("{}\t{}\t{}\t{}".format(phrase,target,index,result))
160                 file2.write("\n")
161                 texte = file.readline()
162 #*****
163 # Cette fonction prend en argument un corpus victor et en sortie un fichier avec les candidats
164 # classés du plus facile au plus compliqué
165 #*****
166 def main_longueur(victor_corpus,vic):
167     with open(victor_corpus,"r") as file:
168         with open(vic,"w") as file2:
169             texte = file.readline()
170             dictionnaire = {}
171             while texte:
172                 texte_cut = texte.strip().split("\t")
173                 phrase = texte_cut[0]
174                 target = texte_cut[1]
175                 index = int(texte_cut[2])
176                 try:
177                     candidat_cut = texte_cut[3]
178                 except:
179                     candidat_cut = ""
180                 cdt = candidat_cut.split()
181                 for candidat in cdt:
182                     long = prescripteur_longueur(candidat)
183                     if target not in dictionnaire:
184                         dictionnaire[target] = []
185                     dictionnaire[target].append((candidat,long))
186                 liste = test_dico(dictionnaire,target)
187                 result = conversion_string(liste)
188                 file2.write("{}\t{}\t{}\t{}".format(phrase,target,index,result))
189                 file2.write("\n")
190                 texte = file.readline()
191 #*****
192 # Cette fonction permet renvoyer une liste de mots classés du plus grand au plus petit
193 # Cette fonction est utilisé pour les sens et pour le fréquences
194 #*****
195 def test_dico_2(dico,target):
196     dicot = {}
197     liste = []
198     result = {}

```

```

199     for key,val in dico.items():
200         if key == target:
201             for i,j in val:
202                 if i not in dicot:
203                     dicot[i] = []
204                 try:
205                     dicot[i].append(float(j))
206                 except:
207                     dicot[i].append(j)
208     for key,value in dicot.items():
209         if value == [None]:
210             value = [0]
211     result[key] = value
212     for w in sorted(result, key= result.get, reverse=True):
213         liste.append(w)
214     return liste
215
216 *****
217 # Cette fonction, de la même manière que la précédente, renvoie une liste de mots classés
218 # du plus petit au plus grand(pour le prescripteur de la longueur)
219 *****
220 def test_dico(dico,target):
221     dicot = {}
222     liste = []
223     for key,val in dico.items():
224         if key == target:
225             for i,j in val:
226                 if i not in dicot:
227                     dicot[i] = []
228                 dicot[i].append(j)
229     for w in sorted(dicot, key=dicot.get, reverse=False):
230         liste.append(w)
231     return liste

```