FUNDAÇÃO CAED UFJF

ANA CAROLINA DOS SANTOS OVÍDIO

04/03/2022

DOCUMENTAÇÃO JUPYTER

Construção de um algoritmo ligado ao Processamento de Linguagem Natural

TROCA ADJETIVO

Modifica adjetivo por um sinônimo ou por um antônimo

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Instalações necessárias antes de utilizar este módulo.

```
! pip install nltk
nltk.download()
! pip install deep_translator
```

In [15]:
```
import nltk
from nltk.corpus import wordnet as wn
from random import choices
from deep_translator import GoogleTranslator
```

In [10]:
```
sentenca = "A menina é inteligente"
```

Tradução da sentença em inglês

Por que traduzir?

```
Porque a maioria das bilbiotecas para tokenizar e para encontrar a morfologia são associadas
à lingua inglesa. Isso otimiza o código, evitando que ele trave dependendo do computador.
```

In [11]:
```
traduzir_ingles  = GoogleTranslator(source='portuguese', target='english').translate(sentenca)
traduzir_ingles
```

Out[11]:
```
'the girl is smart'
```

Tokenização

In [12]:
```
tokens = nltk.word_tokenize(traduzir_ingles)
#Lista de tuplas cujos elementos são respectivamente a palavra e sua morfologia
tagged = nltk.pos_tag(tokens)
tagged
```

Out[12]:
```
[('the', 'DT'), ('girl', 'NN'), ('is', 'VBZ'), ('smart', 'JJ')]
```

In [13]:
```
for tag in tagged:
    if tag[1]== 'JJ':
        adjetivo = tag[0]
        break
    else:
        adjetivo = None
print(adjetivo)
```

```
smart
```

Encontra os sinônimos e antônimos

In [16]:
```
synonyms = []
antonyms = []

for syn in wn.synsets(adjetivo):
    for lemma in syn.lemmas():
        synonyms.append(lemma.name())
        if lemma.antonyms():
            antonyms.append(lemma.antonyms()[0].name())
```

```
for palavra in synonyms:
    if palavra == adjetivo:
        synonyms.remove(palavra)

for palavra in antonyms:
    if palavra == adjetivo:
        antonyms.remove(palavra)
```

In [17]:
```
print(f'Sinônimos: {synonyms}')
print(f'\nAntônimos: {antonyms}')
```

```
Sinônimos: ['smarting', 'smartness', 'ache', 'hurt', 'chic', 'voguish', 'bright', 'fresh', 'impertinent', 'im
pudent', 'overbold', 'saucy', 'sassy', 'wise', 'smart']

Antônimos: ['stupid']
```

In [18]:
```
from random import choices

tipo_troca = choices(['synonyms', 'antonyms'])
tipo_troca

if (len(antonyms) == 0) and (len(synonyms) != 0):
    novo_adjetivo = choices(synonyms)
elif (len(synonyms) == 0) and (len(antonyms) != 0):
    novo_adjetivo = choices(antonyms)
elif (len(antonyms) == 0) and (len(synonyms)!=0):
    novo_adjetivo = None
elif tipo_troca == 'synonyms':
    novo_adjetivo = choices(synonyms)
else:
    novo_adjetivo = choices(antonyms)


print(f'Adjetivo selecionado: {novo_adjetivo[0]}')

nova_sentenca = traduzir_ingles.replace(adjetivo,novo_adjetivo[0])

nova_sentenca_pt = GoogleTranslator(source='english', target='portuguese').translate(nova_sentenca)
```

```
Adjetivo selecionado: stupid
```

Resultado

In [19]:
```
nova_sentenca_pt
```

Out[19]:
```
'a menina é burra'
```

TROCA GÊNERO

Modifica o gênero da sentença

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Instalações necessárias antes de rodar o código

```
!pip install nltk
!pip install selenium --user
```

Observação: *--user* é somente em caso do Jupyter não reconhecer alguns comandos do seu navegador.

In [1]:
```
import nltk
```

In [3]:
```
sentenca = "Aqueles amigos eram engraçados"

#Garantir que a frase sempre começe com letra maiúscula.
# Não vale a pena utilizar o replace.
lista_sentenca = list(sentenca)
lista_sentenca[0] = lista_sentenca[0].upper()
sentenca = ''.join(lista_sentenca)
sentenca
```

Out[3]:
```
'Aqueles amigos eram engraçados'
```

Tokenização

tokens = Uma lista cujos elementos são as palavras da string *sentenca*

In [6]:
```
tokens = nltk.word_tokenize(sentenca)
if '.' in tokens:
    tokens.remove('.')
tokens
```

Out[6]:
```
['Aqueles', 'amigos', 'eram', 'engraçados']
```

Descobre qual gênero a frase atual foi empregada

In [7]:
```
pronomes_artigos_femininos = ['A','Uma','Alguma','Aquela','Umas','Algumas','Aquelas']
pronomes_artigos_masculinos = ['O', 'Um', 'Algum', 'Aquele', 'Uns', 'Alguns', 'Aqueles']

if tokens[0] in pronomes_artigos_femininos:
    genero = 'feminino'
elif tokens[0] in pronomes_artigos_masculinos:
    genero = 'masculino'
else:
    genero = None

display(genero)
```

```
'masculino'
```

Classe responsável por trocar o gênero

Função: ***init***

Função que define as variáveis globais de todas as funções pertencentes à classe.

```
self.tokens = lista com as palavras da string sentenca

self.genero = gênero inicial

self.inicio_fem = lista com artigos e pronomes femininos que estarão no inicio da frase.

self.inicio_masc = lista com artigos e pronomes masculinos que estarão no inicio da frase.
```

Função: *modifica_elementos_sing*

Caso a sentenca inicial esteja no singular:

```
Verifica qual o gênero e estabelece três principios de mudança:

1- Terminações or/ ora -- (cantor/cantora)
2- Terminações com a/ o desconsiderando o primeiro caso (amiga/ amigo)
3- Substantivos que equivalem tanto para feminino e para masculino.
```

Limitação: Caso o substantivo seja homem/mulher, cão/cadela ect o algoritmo não traduz.

Uma solução seria a implementação de uma automação web, mas exige um tempo maior de implementação uma vez que a biblioteca *selenium.py* -responsável pela automação- depende do navegador padrão de cada computador.

```
 Após substituir o substantivo, o algoritmo mantém o verbo e modifica o adjetivo.
```

Função: *modifica_elementos_plural*

Mesmo processo que a função *modifica_elementos_sing*. No entanto, esta função é específica para sentenças no plural.

In [9]:
```
class TrocaGenero ():
    def __init__(self, genero, tokens):

        self.tokens = tokens
        self.genero = genero
        self.inicio_fem = ['A','Uma','Alguma','Aquela','Umas','Algumas','Aquelas']
        self.inicio_masc = ['O', 'Um', 'Algum', 'Aquele', 'Uns', 'Alguns', 'Aqueles']


    def modifica_elementos_sing(self):
        troca_genero = []

        if self.genero == 'feminino':
            pos_artigo = self.inicio_fem.index(self.tokens[0])
            troca_genero.append(self.inicio_masc[pos_artigo])

            #Trocar o substantivo
            if 'ora' in self.tokens[1]:
                 troca_genero.append(self.tokens[1].replace('ora', 'or'))
            # Não há problema esta verificação, pois a string 'ora' já foi verificada
            elif 'a' in self.tokens[1][-1]:
                tamanho_sub = len(self.tokens[1])
                troca_genero.append(self.tokens[1][:tamanho_sub-1] + 'o')
            elif ('ente', 'esta', 'ante'):
                troca_genero.append(self.tokens[1])

            # Insere verbo
            troca_genero.append(self.tokens[2])

            #Modifica adjetivo
            if self.tokens[3][-1] =='a':
                tamanho_adj = len(self.tokens[3])
                troca_genero.append(self.tokens[3][:tamanho_adj-1] + 'o')
            else:
                troca_genero.append(self.tokens[3])

            troca_genero.append('.')
            texto_modificado = ' '.join(troca_genero)

        else:
            pos_artigo = self.inicio_masc.index(self.tokens[0])
```

```
            troca_genero.append(self.inicio_fem[pos_artigo])

            #Trocar o substantivo
            if 'or' in self.tokens[1]:
                 troca_genero.append(self.tokens[1].replace('or', 'ora'))
            # Não há problema esta verificação, pois a string 'ora' já foi verificada
            elif 'o' in self.tokens[1][-1]:
                tamanho_sub = len(self.tokens[1])
                troca_genero.append(self.tokens[1][:tamanho_sub-1] + 'a')
            elif ('ente', 'esta', 'ante'):
                troca_genero.append(self.tokens[1])

            # Insere verbo
            troca_genero.append(self.tokens[2])

            #Modifica adjetivo
            if self.tokens[3][-1] =='o':
                tamanho_adj = len(self.tokens[3])
                troca_genero.append(self.tokens[3][:tamanho_adj-1] + 'a')
            else:
                troca_genero.append(self.tokens[3])

            texto_modificado = ' '.join(troca_genero)
        return texto_modificado

    #-------------------------------------------------------------------------------

    def modifica_elementos_plural(self):
        troca_genero = []

        if self.genero == 'feminino':
            pos_artigo = self.inicio_fem.index(self.tokens[0])
            troca_genero.append(self.inicio_masc[pos_artigo])

            #Trocar o substantivo
            if self.tokens[1][-4:] == 'oras':
                 troca_genero.append(self.tokens[1].replace('oras', 'ores'))

            # Não há problema esta verificação, pois a string 'ora' já foi verificada
            elif self.tokens[1][-2:] == 'as':
                tamanho_sub = len(self.tokens[1])
                troca_genero.append(self.tokens[1][:tamanho_sub-2] + 'os')

            elif self.tokens[1][-5:] in ['entes', 'estas', 'antes']:
                troca_genero.append(self.tokens[1])

            # Insere verbo
            troca_genero.append(self.tokens[2])

            #Modifica adjetivo
            if self.tokens[3][-2:] =='as':
                tamanho_adj = len(self.tokens[3])
                troca_genero.append(self.tokens[3][:tamanho_adj-2] + 'os')
            else:
                troca_genero.append(self.tokens[3])
            texto_modificado = ' '.join(troca_genero)

        #------
        else:
            pos_artigo = self.inicio_masc.index(self.tokens[0])
            troca_genero.append(self.inicio_fem[pos_artigo])

            #Trocar o substantivo
            if self.tokens[1][-4:] == 'ores':
                 troca_genero.append(self.tokens[1].replace('ores', 'oras'))
            elif self.tokens[1][-2:] == 'os':
                tamanho_sub = len(self.tokens[1])
                troca_genero.append(self.tokens[1][:tamanho_sub-2] + 'as')
            elif self.tokens[1][-5:] in ['entes', 'estas', 'antes']:
                troca_genero.append(self.tokens[1])

            # Insere verbo
            troca_genero.append(self.tokens[2])

            #Modifica adjetivo
            if self.tokens[3][-2:] =='os':
                tamanho_adj = len(self.tokens[3])
                troca_genero.append(self.tokens[3][:tamanho_adj-2] + 'as')
            else:
                troca_genero.append(self.tokens[3])
```

```
            texto_modificado = ' '.join(troca_genero)

        return texto_modificado
```

Verifica a numeração da sentença e chama as respectivas funções da classe acima

In [10]:
```
tg = TrocaGenero(genero, tokens)

if tokens[0][-1] != 's':
    numeracao = 'singular'
else:
    numeracao = 'plural'

if numeracao == 'singular':
    nova_sentenca = tg.modifica_elementos_sing()
else:
    nova_sentenca = tg.modifica_elementos_plural()
nova_sentenca = nova_sentenca + '.'
nova_sentenca
```

Out[10]:
```
'Aquelas amigas eram engraçadas.'
```

SUJEITO INDETERMINADO

Permutação entre orações com sujeito indeterminado

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Instalações necessárias antes de utilizar este módulo.

```
! pip install nltk
! pip install deep_translator
```

In [1]:
```
import nltk
from nltk.corpus import brown
from itertools import permutations
from deep_translator import GoogleTranslator
```

Tradução da sentença em inglês

Por que traduzir?

```
Porque a maioria das bilbiotecas para tokenizar e para encontrar a morfologia são associadas
à lingua inglesa. Isso otimiza o código, evitando que ele trave dependendo do computador.
```

In [2]:
```
sentenca = 'Trovejou nessa manhã'

traduzir_ingles  = GoogleTranslator(source='portuguese', target='english').translate(sentenca)
print(f'Tradução para o inglês: {traduzir_ingles}')
```

```
Tradução para o inglês: It thundered this morning
```

Tokenização

In [3]:
```
tokens = nltk.word_tokenize(traduzir_ingles)
if '.' in tokens:
    tokens.remove('.')
#Lista de tuplas cujos elementos são respectivamente a palavra e sua morfologia
tagged = nltk.pos_tag(tokens)
display(tagged)
```

```
[('It', 'PRP'), ('thundered', 'VBD'), ('this', 'DT'), ('morning', 'NN')]
```

Separa em duas orações

Uma vez que verbo auxiliar não deve ser separado do verbo principal. Como no caso de *estava chovendo*

Observações:

- Como não há sujeito, uma classificação morfológica 'NN' ou 'NNS' representa um advérbio.
- Separar o advérbio do verbo é a principal função.

In [4]:
```
primeira_oracao = []
for i, tag in enumerate(tagged):
    if tag[1][0] == 'N':
        existe_verbo = False
        break
    elif tag[1][0] == 'V':
        existe_verbo = True
        break
    else:
        primeira_oracao.append(tag)
```

```
if existe_verbo == True:
    primeira_oracao.append(tag)
    i = i+1
    while i != len(tagged)-1:
        if tagged[i][1][0] == 'V':
            primeira_oracao.append(tagged[i])
            i = i+1
        else:
            break
    segunda_oracao = tagged[i+1:]
else:
    primeira_oracao.append(tag)
    i = i+1
    while i != len(tagged)-1:
        if (tagged[i][1][0] != 'V') and (tagged[i][1] != 'PRP'):
            primeira_oracao.append(tagged[i])
            i = i+1
        else:
            break
    segunda_oracao = tagged[i:]

for i,tag in enumerate(primeira_oracao):
    primeira_oracao[i] = tag[0]
for i,tag in enumerate(segunda_oracao):
    segunda_oracao[i] = tag[0]
```

Permutação entre a oração que contém o advérbio e aquela que contém o verbo

In [5]:
```
def permutacao_oracoes (oracoes):

    primeira_permutacao = list(permutations(oracoes))

    #Tranformar tuplas em lista para permitir modificação
    permutacoes = []
    for tupla in primeira_permutacao:
        permutacoes.append(list(tupla))
    permutacoes_concatenadas = []

    #Concatenar as strings
    for permutacao in permutacoes:
        if permutacao[0][0] == permutacao[0][0].upper():
            pass
        else:
            primeira_string = list(permutacao[0])
            primeira_string[0] = primeira_string[0].upper()
            permutacao[0] = ''.join(primeira_string)

            segunda_string = list(permutacao[1])
            segunda_string[0] = segunda_string[0].lower()
            permutacao[1] = ''.join(segunda_string)

        permutacoes_concatenadas.append(' '.join(permutacao))

    return permutacoes_concatenadas
```

In [6]:
```
primeira_oracao = ' '.join(primeira_oracao)
segunda_oracao = ' '.join(segunda_oracao)

print(f'Oração 1: {primeira_oracao}')
print(f'Oração 2: {segunda_oracao}')

oracoes = []
oracoes.append(primeira_oracao)
oracoes.append(segunda_oracao)

permutacoes = permutacao_oracoes(oracoes)
```

```
Oração 1: It thundered
Oração 2: morning
```

Resultado

In [7]:
```
possibilidades_frase = []
for permutacao in permutacoes:
    possibilidades_frase.append(GoogleTranslator(source='english', target='portuguese').translate(permutacao)
for i in possibilidades_frase:
    print(i)
```

```
trovejou de manhã
Manhã trovejou
```

SUJEITO EXPLÍCITO

Permutação entre orações com sujeito explicito

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Instalações necessárias antes de utilizar este módulo.

```
! pip install nltk
nltk.download()
! pip install deep_translator
```

In [8]:
```
import nltk
from itertools import permutations
from deep_translator import GoogleTranslator
```

Tradução da sentença em inglês

Por que traduzir?

```
Porque a maioria das bilbiotecas para tokenizar e para encontrar a morfologia são associadas
à lingua inglesa. Isso otimiza o código, evitando que ele trave dependendo do computador.
```

In [9]:
```
sentenca = 'A Segunda Guerra Mundial, conflito militar entre 1939 a 1945, envolveu as maiores potências da ép

traduzir_ingles  = GoogleTranslator(source='portuguese', target='english').translate(sentenca)
print(f'Tradução para o inglês: {traduzir_ingles}')
```

```
Tradução para o inglês: The Second World War, a military conflict between 1939 and 1945, involved the greates
t powers of the time.
```

Tokenização

In [10]:
```
tokens = nltk.word_tokenize(traduzir_ingles)
if '.' in tokens:
    tokens.remove('.')
#Lista de tuplas cujos elementos são respectivamente a palavra e sua morfologia
tagged = nltk.pos_tag(tokens)
display(tagged)
```

```
[('The', 'DT'),
 ('Second', 'JJ'),
 ('World', 'NNP'),
 ('War', 'NNP'),
 (',', ','),
 ('a', 'DT'),
 ('military', 'JJ'),
 ('conflict', 'NN'),
 ('between', 'IN'),
 ('1939', 'CD'),
 ('and', 'CC'),
 ('1945', 'CD'),
 (',', ','),
 ('involved', 'VBD'),
 ('the', 'DT'),
 ('greatest', 'JJS'),
 ('powers', 'NNS'),
 ('of', 'IN'),
 ('the', 'DT'),
 ('time', 'NN')]
```

Primeira função a ser chamada

Separa sujeito (inclui o aposto) do predicado.

Posteriormente, separa o sujeito do aposto caso exista.

In [4]:
```
def separa_elementos_oracao (tagged):

    sujeito = []
    predicado = []

    #Separar o sujeito do predicado
    for i,tag in enumerate(tagged):
        if tag[1][0]== 'V':
            break
        else:
            sujeito.append(tag[0])

    for j in range(i,len(tagged)):
        predicado.append(tagged[j][0])
    predicado_concatenado = []
    predicado_concatenado.append(' '.join(predicado))

    #Separar o sujeito principal do aposto
    aposto = []
    sujeito_principal = []

    qnt_virgulas = 0
    posicao_virgulas = []
    for i,tag in enumerate(tagged):
        if tag[0] == ',':
            qnt_virgulas = qnt_virgulas + 1
            posicao_virgulas.append(i)

    if len(posicao_virgulas) == 1:
        aposto = sujeito[:posicao_virgulas[0]]
        sujeito_principal = sujeito[posicao_virgulas[0]+1:]
    elif len(posicao_virgulas) == 2:
        aposto = sujeito[posicao_virgulas[0]+1:posicao_virgulas[1]]
        sujeito_principal = sujeito[:posicao_virgulas[0]]+sujeito[posicao_virgulas[1]+1:]
    else:
        sujeito_principal = sujeito
    suj = [] #sujeito em string
    aposto_suj = [] #aposto em string

    suj.append(' '.join(sujeito_principal))
    if len(aposto) != 0:
        aposto_suj.append(' '.join(aposto))
    else:
        aposto_suj.append(None)

    return suj[0], aposto_suj[0], predicado_concatenado
```

Permutação entre o sujeito e o aposto

O aposto da uma característica ao sujeito, logo a frase não tem sentido caso exista a permutação: sujeito + verbo + aposto OU aposto + verbo + sujeito

In [5]:
```
def permutacao_sujeito_aposto(suj, aposto=None):

    if aposto != None:
        primeira_permutacao = list(permutations([suj, aposto]))

        inserir_virgula = []
        permutacoes_string = []

        #Tranformar tuplas em lista para permitir modificação
        permutacoes = []
        for tupla in primeira_permutacao:
            permutacoes.append(list(tupla))

        for i, permutacao in enumerate(permutacoes):
            #A primeira sentenca da permutação começa com letra maiúscula
            if permutacao[0][0] == permutacao[0][0].upper():
                pass
            else:
                lista_palavras = list(permutacao[0])
                lista_palavras[0] = lista_palavras[0].upper()
                permutacao[0] = ''.join(lista_palavras)
```

```
            #A segunda sentenca da permutação começa com letra maiúscula
            if permutacao[1][0] == permutacao[1][0].upper():
                lista_palavras = list(permutacao[1])
                lista_palavras[0] = lista_palavras[0].lower()
                permutacao[1] = ''.join(lista_palavras)
            else:
                pass

        # Concatenar strings permutadas
        permutacoes_concatenadas = []
        for i in range(len(permutacoes)):
            permutacoes_concatenadas.append(' '.join(permutacoes[i]))

    else:
        permutacoes_concatenadas = [suj]

    return permutacoes_concatenadas
```

Permutação entre as permutações da função anterior com o predicado

In [6]:
```
def permutacao_sujeito_predicado(sujeitos, predicado):

    primeira_permutacao = []
    for sujeito in sujeitos:
        primeira_permutacao.append(list(permutations([sujeito, predicado[0]])))

    #Tranformar tuplas em lista para permitir modificação
    permutacoes = []
    for lista in primeira_permutacao:
        for tupla in lista:
            permutacoes.append(list(tupla))

        for i, permutacao in enumerate(permutacoes):
            #A primeira sentenca da permutação começa com letra maiúscula
            if permutacao[0][0] == permutacao[0][0].upper():
                pass
            else:
                lista_palavras = list(permutacao[0])
                lista_palavras[0] = lista_palavras[0].upper()
                permutacao[0] = ''.join(lista_palavras)
            #A segunda sentenca da permutação começa com letra maiúscula
            if permutacao[1][0] == permutacao[1][0].upper():
                lista_palavras = list(permutacao[1])
                lista_palavras[0] = lista_palavras[0].lower()
                permutacao[1] = ''.join(lista_palavras)
            else:
                pass

        # Concatenar strings permutadas
        permutacoes_concatenadas = []
        for i in range(len(permutacoes)):

            if permutacoes[i][-1][-1] == ',':
                frase_com_virgula = list(permutacoes[i][-1])
                frase_com_virgula[-1] = '.'
                permutacoes[i][-1] = ''.join(frase_com_virgula)
            permutacoes_concatenadas.append(' '.join(permutacoes[i]))

    return permutacoes_concatenadas
```

In [11]:
```
#Nesse caso não há sujeito indeterminado tampouco oculto.

sujeito, aposto, predicado = separa_elementos_oracao(tagged)
print(f'Sujeito: {sujeito} - Aposto: {aposto}')

if aposto != None:
    permt_suj_apost = permutacao_sujeito_aposto(sujeito, aposto=aposto)

else:
    permt_suj_apost = permutacao_sujeito_aposto(sujeito)

permutacoes = permutacao_sujeito_predicado(permt_suj_apost, predicado)
```

```
Sujeito: The Second World War - Aposto: a military conflict between 1939 and 1945
```

Resultado

In [12]:
```
for permutacao in permutacoes:
    if permutacao[-1]!= '.':
        permutacao = permutacao + '.'
    permutacao = GoogleTranslator(source='english', target='portuguese').translate(permutacao)
    print(permutacao)
```

```
A Segunda Guerra Mundial um conflito militar entre 1939 e 1945 envolveu as maiores potências da época.
Envolveu as maiores potências da época a Segunda Guerra Mundial um conflito militar entre 1939 e 1945.
Um conflito militar entre 1939 e 1945 a Segunda Guerra Mundial envolveu as maiores potências da época.
Envolveu as maiores potências da época um conflito militar entre 1939 e 1945 a Segunda Guerra Mundial.
```

VOZ ATIVA E VOZ PASSIVA

Troca voz ativa por voz passiva ou vice- versa

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Observação

Por que não houve divisão em classes neste código?

A *orientação objeto* no Jupyter é complicada e, como o notebook está sendo utilizando apenas para exemplificação do processamento de linguagem natural, optou- se por deixar os comentários em cima de cada processo do programa.

Instalações necessárias antes de utilizar este módulo.

```
! pip install pattern
! pip install pyinflect
! pip install deep_translator
```

In [1]:
```
import nltk
import pyinflect
import spacy
from deep_translator import GoogleTranslator
```

In [2]:
```
nltk.download()
spacy.cli.download("en_core_web_sm")
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

Mudança da voz verbal em uma sentença.

```
    1) Traduzir para o inglês uma vez que há funções mias otmizadas no NLTK   para essa
linguagem.
    2) Tokenização
    3) Verificar o tempo verbal
    4) Realizar a modificação
    5) Traduzir para o português
```

Funções responsáveis por transformar a voz

As funções serão chamadas a realização da tokenixação do texto em inglês

In [3]:
```
def modificar_para_ativa (objeto, sujeito, tagged, posicao_verbo, posicao_objeto,
                  posicao_sujeito, verbo_passado):

    novo_texto = []

    #Responsável por realizar as trocas
    for i in range (len(tagged)):
        if i == posicao_verbo:
            novo_texto.append(verbo_passado)
        elif i == posicao_sujeito:
            novo_texto.append(objeto)
        elif i == posicao_objeto:
            novo_texto.append(sujeito)
        else:
            novo_texto.append(tagged[i][0])
```

```
    #Remove verbo no participio passado
    novo_texto.pop(posicao_verbo+1)

    # Excluir o by pois essa preposição não existe na voz ativa
    novo_texto.remove('by')

    return novo_texto
```

In [4]:
```
def modificar_para_passiva (objeto, sujeito, tagged, posicao_verbo, posicao_objeto,
              posicao_sujeito, verbo_participio_passado):

    novo_texo = []
    for tag in tagged:
        if tag[0] == objeto:
            tag_objeto = tag[1]
    for i in range (len(tagged)):

        if i == posicao_verbo:
            # Valida qual conjugação do verbo to be no passado se adequa ao objeto
            # O objeto será o novo sujeito
            if ((tag_objeto == 'NNS') or
                (tag_objeto in ['they', 'They', 'we', 'We', 'you', 'You'])):
                    novo_texo.append('were')
            else:
                    novo_texo.append('was')
            novo_texo.append(verbo_participio_passado)

        elif i == posicao_sujeito:
            novo_texo.append(objeto)
        elif i == posicao_objeto:
            #Adicionar a preposição by
            novo_texo.insert(i,'by')
            novo_texo.append(sujeito)
        else:
            novo_texo.append(tagged[i][0])

    return novo_texo
```

Início do processo de mudança de voz

Tradução da sentença e tratamento dela

In [5]:
```
sentenca = "O objeto foi pego, no chão, pelo cachorro"

if '.' in sentenca:
    sentenca = sentenca.replace('.', '')

traduzir_ingles  = GoogleTranslator(source='portuguese', target='english').translate(sentenca)
print(f'Tradução para o inglês: {traduzir_ingles}')
```

```
Tradução para o inglês: The object was picked up, on the ground, by the dog
```

Verificação se há um trecho entre vírgulas separando sujeito e o verbo

In [6]:
```
# Caso não exista um trecho entre vírgulas na sentença
palavra_antes_verbo = None

tokens = nltk.word_tokenize(traduzir_ingles)
tagged = nltk.pos_tag(tokens)
for i,tag in enumerate(tagged):
    if tag[0] == ',':
        palavra_antes_verbo = tagged[i-1][0]
        break

# Armazenar em uma string o trecho entre vírgula
# Retirar esse trecho da sentenca em inglês por enquanto
if palavra_antes_verbo != None:
    for i in range(len(traduzir_ingles)):
        if traduzir_ingles[i] == ',':
            posicao_virgula1 = i
            break

    for i in range (posicao_virgula1+1, len(traduzir_ingles)):
        if traduzir_ingles[i] == ',':
```

```
            posicao_virgula2 = i
            break

    entre_virgulas = traduzir_ingles[posicao_virgula1:posicao_virgula2+1]
    traduzir_ingles = traduzir_ingles.replace(entre_virgulas,"")
    print(f'Tradução sem o trecho entre vírgulas: {traduzir_ingles}')
```

```
Tradução sem o trecho entre vírgulas: The object was picked up by the dog
```

Tokenização

In [7]:
```
tokens = nltk.word_tokenize(traduzir_ingles)
#Lista de tuplas cujos elementos são respectivamente a palavra e sua morfologia
tagged = nltk.pos_tag(tokens)
display(tagged)
```

```
[('The', 'DT'),
 ('object', 'NN'),
 ('was', 'VBD'),
 ('picked', 'VBN'),
 ('up', 'RP'),
 ('by', 'IN'),
 ('the', 'DT'),
 ('dog', 'NN')]
```

Identificar o sujeito e o objeto + Verificação se a voz é passiva ou ativa

In [8]:
```
# Encontra o objeto da frase
for i in range(len(tagged)-1, -1, -1):
    if (tagged[i][1] == 'NN') or (tagged[i][1] == 'NNS'):
        objeto = tagged[i][0]
        posicao_objeto = i
        break

# Encontra o sujeito da frase
for i,tag in enumerate(tagged):
    if (tag[1] == 'NN') or (tag[1] == 'NNS'):
        sujeito = tag[0]
        posicao_sujeito = i
        break

#Encontra a posição do verbo
for i in range(len(tagged)):
    #Verifica se o token é um verbo (independente do tempo)
    if tagged[i][1][0] == 'V':
        posicao_verbo = i
        break

#Necessidade de tranformar o texto em um objeto spacy.load para trabalhar com tempos verbais
nlp = spacy.load("en_core_web_sm")
doc_dep = nlp(traduzir_ingles)
```

Verifica o tipo de voz a frase atual está empregada

In [9]:
```
verbo_passado = None
verbo_participio_passado = None

# Verifica se a voz é passiva (verb to be in the pass + verb in the past participle)
if (tagged[posicao_verbo][1] == 'VBD') and (tagged[posicao_verbo+1][1] == 'VBN'):
    for i in range(1,len(doc_dep)):
        token = doc_dep[i-1]
        proximo_token = doc_dep[i]
        if (token.tag_ == 'VBD')  and (proximo_token.tag_ == 'VBN'):
            verbo_passado = proximo_token._.inflect("VBD")

    #Representa uma lista com a sentenca ordenada para outra voz
    traducao_para_portugues = modificar_para_ativa(objeto, sujeito,tagged, posicao_verbo,
                                    posicao_objeto,
                                    posicao_sujeito, verbo_passado)


# Senão a voz é ativa (verb in the past)
else:
    for i in range(len(doc_dep)):
        token = doc_dep[i]
        if token.tag_ == 'VBD':
            verbo_participio_passado = token._.inflect("VBN")
```

```
    #Representa uma lista com a sentenca ordenada para outra voz
    traducao_para_portugues = modificar_para_passiva(objeto, sujeito,tagged, posicao_verbo,
                                    posicao_objeto,
                                    posicao_sujeito, verbo_participio_passado)

display(f'Verbo no passado: {verbo_passado}')
display(f'Verbo no participio passado: {verbo_participio_passado}')
print(traducao_para_portugues)
```

```
'Verbo no passado: picked'
'Verbo no participio passado: None'
['The', 'dog', 'picked', 'up', 'the', 'object']
```

Tranformar a lista de palavras em uma string

A partir de agora, caso existisse vírgula entre o sujeito e o verbo, a string é retificada.

In [10]:
```
if palavra_antes_verbo != None:
    #Uma lista com todos os elementos da string
    entre_virgulas = entre_virgulas.split()

    indice_palavra_antes = traducao_para_portugues.index(palavra_antes_verbo)

    for i in range(len(entre_virgulas)):
        traducao_para_portugues.insert(indice_palavra_antes+1+i, entre_virgulas[i])

    #Retirar a segunda vírgula caso entre_virgulas estiver no final da frase na modificação.

    if entre_virgulas[-1] == traducao_para_portugues[-1]:
        ultimo_elemento = entre_virgulas[-1]
        ultimo_elemento = ultimo_elemento.replace(',', "")
        traducao_para_portugues[-1] = ultimo_elemento
    else:
        pass

    traducao_para_portugues = " ".join(traducao_para_portugues)

else:
    traducao_para_portugues = " ".join(traducao_para_portugues)

traducao_para_portugues
```

Out[10]:
```
'The dog picked up , on the ground, the object'
```

Retorno para o português

In [11]:
```
traducao_para_portugues = traducao_para_portugues + '.'
nova_sentenca = GoogleTranslator(source='english', target='portuguese').translate(traducao_para_portugues)
sentenca = sentenca + '.'
print(f'Sentença para ser modificada: {sentenca}')
print(f'Sentença modificada: {nova_sentenca}')
```

```
Sentença para ser modificada: O objeto foi pego, no chão, pelo cachorro.
Sentença modificada: O cachorro pegou, no chão, o objeto.
```

PALAVRAS CANÔNICAS

Verifica a existência de palavras canônicas

COMO RODAR?

Dentro de cada célula, apertar *Ctrl+Enter* ou clicar em *Run* - ícone superior da página.

Observação 1:

- Rodar as células em ordem. A última célula terá o resultado que deseja.

Caso queira rodar todas as células de uma vez: cell -> Run All

Observação 2:

- Qualquer modificação em uma célula, todas devem ser rodadas novamente e em ordem.

Instalações necessárias antes de utilizar este módulo.

```
! pip install nltk
```

In [2]:
```
import nltk
```

In [3]:
```
sentenca = 'A menina estava com a cabeça na janela'
```

In [4]:
```
lista_vogais = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
```

Tokenização

In [5]:
```
tagged_sents = nltk.corpus.mac_morpho.tagged_sents()
tokens = nltk.word_tokenize(sentenca)
if '.' in tokens:
    tokens.remove('.')

tokens
```

Out[5]:
```
['A', 'menina', 'estava', 'com', 'a', 'cabeça', 'na', 'janela']
```

Raciocínio

- Todas as sílabas devem ter duas letras, logo o tamanho da palavra deve ser divisível por 2.
- Todas as sílabas devem ser CV.

In [8]:
```
palavras_canonicas = []

for token in tokens:
    count = 0
    tamanho_palavra = len(token)

    if tamanho_palavra%2 == 0:
        silabas = []
        for i in range(0, len(token), 2):
            j = i+2
            silabas.append(token[i:j])
            i=j

        for silaba in silabas:
            if ((silaba[0] not in lista_vogais) and
                (silaba[1] in lista_vogais)):
                count = count + 1
            else:
                break

        if count == tamanho_palavra/2:
            palavras_canonicas.append(token)
    else:
        pass
```

Resultado

In [9]:
```
palavras_canonicas
```

Out[9]:
```
['menina', 'cabeça', 'na', 'janela']
```