«گزارش کار تمرین شماره ۱»

عنوان تمرین	گزارش تکلیف شماره ۱
عنوان درس	مباحث ویژه در تجارت الکترونیک (پردازش زبان طبیعی – NLP)
نام و نام خانوادگی	امیرحسین خانیکی
شماره دانشجویی	۴۰۱۳۶۱۵۰۰۶
راه ارتباطی	۰۹۱۵۵۱۷۶۹۷۶ و t.me/khaniki_ah

۱) سازمان‌دهی فضاهای خالی

برای سازمان‌دهی فضای متنی در متن باید فاصله، تب و یا خط جدید حذف شوند و که بدین منظور از WhitespaceTokenizer استفاده شده است. پیش از آن، برای خواندن محتوای فایل (ها) ابتدا با استفاده از متد open() فایل باز و بعد با استفاده از متد read() محتوای آن خوانش و سپس عملیات سازمان‌دهی برای بین بردن فضاهای خالی اجرا می‌شود. در نهایت همه توکن‌های موجود در متن در یک متغیر لیستی/آرایه‌ای استخراج، ذخیره و به‌وسیله متد join()، مجدد با تبدیل لیست به رشته به شکل یک جمله/عبارت قابل مشاهده است. ایراد متد WhitespaceTokenizer در این است که گاهی علائم نگارشی مستقل را به شکل یک توکن مجزا در نظر می‌گیرد، همانند :) ، و از سوی دیگر داده‌های زبان فارسی را به درستی تحلیل و بررسی نمی‌کند که برای حوزه زبان فارسی بسته PartNLP پیشنهاد می‌گردد.

```
Tokenized Result (en):  ['Hello!', 'Hope', "you're", 'doing', 'well.', 'It', 'is', 'a', 'sample',
nt', ':)', '.']
Tokenized Result (fa):  ['\ufeffس ال م!', 'م ر ا و د ی م ا', 'ب و خ', 'د ی ش آ ب', ':-)', '.', 'ا ی ن', 'ک ی', 'ت ت م
.']
Question-1 Result (en):  Hello! Hope you're doing well. It is a sample short text.This is our 1 s
Question-1 Result (fa):  س ال م ! ا ی د و ا ر م ب و خ ب آ ش ی د :- ) . ا ی ن ک ی ت ت م ه ا ت و ک ن و م ن ه ا ت س.ا ی ن ی ر م ت ۱ م
```

نمونه کد در زیر ارائه شده است:

```python
# Datasets
beanstalk_text_path: str = os.path.join(c_path, "datasets/Beanstalk.txt")
short_sample_en_path: str = os.path.join(c_path,
"datasets/ShortSampleEnglish.txt")
short_sample_fa_path: str = os.path.join(c_path,
"datasets/ShortSamplePersian.txt")
zahak_text_path: str = os.path.join(c_path, "datasets/zahak.txt")
extra_text_path: str = os.path.join(c_path, "datasets/extra.txt")

# Contents
beanstalk_text: str = open(beanstalk_text_path, "r", encoding="utf-8",
errors="ignore").read()
```

```
short_sample_en: str = open(short_sample_en_path, "r", encoding="utf-8",
errors="ignore").read()
short_sample_fa: str = open(short_sample_fa_path, "r", encoding="utf-8",
errors="ignore").read()
zahak_text: str = open(zahak_text_path, "r", encoding="utf-8",
errors="ignore").read()
extra_text: str = open(extra_text_path, "r", encoding="utf-8",
errors="ignore").read()
```

```
# Question 1
wst = tokenize.WhitespaceTokenizer()
tokenized_result_en = wst.tokenize(short_sample_en)
tokenized_result_fa = wst.tokenize(short_sample_fa)
tokenized_beanstalk_text = wst.tokenize(beanstalk_text)
tokenized_zahak_text = wst.tokenize(zahak_text)
tokenized_extra_text = wst.tokenize(extra_text)
print("Tokenized Result (en): ", tokenized_result_en)
print("Tokenized Result (fa): ", tokenized_result_fa)
processed_result_en = q1_result_en = " ".join(tokenized_result_en)
processed_result_fa = q1_result_fa = " ".join(tokenized_result_fa)
processed_beanstalk_text = q1_beanstalk_text = "
".join(tokenized_beanstalk_text)
processed_zahak_text = q1_zahak_text = " ".join(tokenized_zahak_text)
processed_extra_text = q1_extra_text = " ".join(tokenized_extra_text)
print("Question-1 Result (en): ", q1_result_en)
print("Question-1 Result (fa): ", q1_result_fa)
```

۲) یکپارچه‌سازی حروف

در برخی زبان‌های انسانی حروف بزرگ و حروف کوچک نگارشی وجود دارند، از جمله همه زبان‌هایی که از الفبای لاتین (به استثنای Saanich) یا الفبای سیریلیک (روسی، بلغاری، مغولی و برخی دیگر) استفاده می‌کنند. به منظور یکپارچه‌سازی حروف نوشتار/رشته می‌توان آن‌ها را به حالت بزرگ یا کوچک تبدیل کرد که در این بخش تمرکز و هدف بر روی کوچک‌سازی است. بدین منظور از متد lower() استفاده می‌شود.

```
Question-1 Result (en):  Hello! Hope you're doing well. It is a sample short text.This is our 1 st assi
Question-1 Result (fa):  سلام! امیدوارم خوب باشید :-) . این یک متن کوتاه نمونه است.این تمرین ۱ ما است.
Question-2 Result:  hello! hope you're doing well. it is a sample short text.this is our 1 st assignmen
```

کد نوشته شده برای این بخش در زیر ارائه شده است:

```
# Question 2
processed_result_en = q2_result_en = q1_result_en.lower()
processed_beanstalk_text = q2_result_beanstalk_text =
q1_beanstalk_text.lower()
processed_extra_text = q2_result_extra_text = q1_extra_text.lower()
print ("Question-2 Result: ", q2_result_en)
```

۳) استخراج جملات و توکن ها

PunktSentenceTokenizer یک الگوریتم بدون نظارت تشخیص مرز جمله و کلاس انتزاعی برای قطعه‌بندی جملات است که در NLTK ارائه شده است. از سوی دیگر، TreebankWordTokenizer از عبارات منظم (RegExp) برای قطعه کردن (Token) متن مانند TreebankWordTokenizer استفاده می‌کند.

حال در پاسخ به اینکه آیا این روش برای متن کوتاه انگلیسی و متن کوتاه فارسی جملات را به درستی تفکیک کرده است؟ باید گفت، خیر و نمی‌تواند مرزبندی را به درستی تفکیک نماید و دلیل مشکل نیز می‌تواند همراه داشتن علائم و نمادهای نگارشی باشد زیرا است در برخی زبان‌ها همانند انگلیسی علامت نگارشی کاربردهای چند منظوره دارند، همانند استفاده نقطه (.) در برخی پیشوندهای جنسی مانند Mr. و Ms. یا شغلی مانند Dr. و یا نام‌های مختصر. به همین دلیل می‌توان از متدهایی همانند متد word_tokenize استفاده کرد که تقسیم جمله و کلمات (توکن) های مناسبی داشته باشیم.

```
Question-3 Result:
(en)        --> Sentences =  3   , Tokens = 21   , Types =  20  @ ['hello!', "hope you're doing well.", '
ent :) .']
(fa)        --> Sentences =  3   , Tokens = 20   , Types =  19
(beanstalk) --> Sentences =  98  , Tokens = 949  , Types =  295
(zahak)     --> Sentences =  15  , Tokens = 1227 , Types =  557
(extra)     --> Sentences =  1   , Tokens = 34   , Types =  30
```

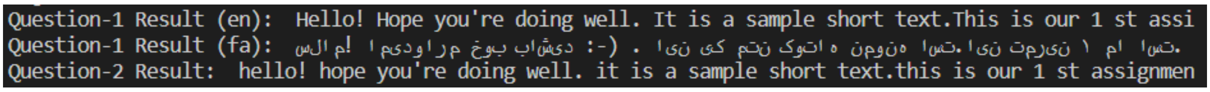
کد نوشته شده برای این بخش در زیر ارائه شده است:

```python
# Question 3
pst = tokenize.PunktSentenceTokenizer()
twt = tokenize.TreebankWordTokenizer()

list_of_sentences_en = pst.tokenize(processed_result_en)
list_of_sentences_fa = pst.tokenize(processed_result_fa)
list_of_sentences_beanstalk = pst.tokenize(processed_beanstalk_text)
list_of_sentences_zahak = pst.tokenize(processed_zahak_text)
list_of_sentences_extra = pst.tokenize(processed_extra_text)

list_of_tokens_tree_en = twt.tokenize(processed_result_en)
list_of_tokens_tree_fa = twt.tokenize(processed_result_fa)
list_of_tokens_tree_beanstalk = twt.tokenize(processed_beanstalk_text)
list_of_tokens_tree_zahak = twt.tokenize(processed_zahak_text)
list_of_tokens_tree_extra = twt.tokenize(processed_extra_text)

list_of_tree_tokens_en = []
list_of_tree_tokens_fa = []
list_of_tree_tokens_beanstalk = []
list_of_tree_tokens_zahak = []
list_of_tree_tokens_extra = []

for i in list_of_sentences_en:
    list_of_tree_tokens_en.append(twt.tokenize(i))
for i in list_of_sentences_fa:
```

```
    list_of_tree_tokens_fa.append(twt.tokenize(i))
for i in list_of_sentences_beanstalk:
    list_of_tree_tokens_beanstalk.append(twt.tokenize(i))
for i in list_of_sentences_zahak:
    list_of_tree_tokens_zahak.append(twt.tokenize(i))
for i in list_of_sentences_extra:
    list_of_tree_tokens_extra.append(twt.tokenize(i))

list_of_tree_tokenz_en = []
list_of_tree_tokenz_fa = []
list_of_tree_tokenz_beanstalk = []
list_of_tree_tokenz_zahak = []
list_of_tree_tokenz_extra = []

for i in list_of_tree_tokens_en:
    for j in i:
        list_of_tree_tokenz_en.append(j)
for i in list_of_tree_tokens_fa:
    for j in i:
        list_of_tree_tokenz_fa.append(j)

for i in list_of_tree_tokens_beanstalk:
    for j in i:
        list_of_tree_tokenz_beanstalk.append(j)

for i in list_of_tree_tokens_zahak:
    for j in i:
        list_of_tree_tokenz_zahak.append(j)
for i in list_of_tree_tokens_extra:
    for j in i:
        list_of_tree_tokenz_extra.append(j)


print ("Question-3 Result:")
print ("(en)        --> Sentences = ", len(list_of_sentences_en), " , Tokens
=", len(list_of_tokens_tree_en), " , Types = ",
len(list(np.unique(list_of_tree_tokenz_en))), " @", list_of_sentences_en)
print ("(fa)        --> Sentences = ", len(list_of_sentences_fa), " , Tokens
=", len(list_of_tokens_tree_fa), " , Types = ",
len(list(np.unique(list_of_tree_tokenz_fa))),)
print ("(beanstalk) --> Sentences = ", len(list_of_sentences_beanstalk), " ,
Tokens =", len(list_of_tokens_tree_beanstalk), " , Types = ",
len(list(np.unique(list_of_tree_tokenz_beanstalk))), )
print ("(zahak)     --> Sentences = ", len(list_of_sentences_zahak), " ,
Tokens =", len(list_of_tokens_tree_zahak), " , Types = ",
len(list(np.unique(list_of_tree_tokenz_zahak))), )
print ("(extra)     --> Sentences = ", len(list_of_sentences_extra), " ,
Tokens =", len(list_of_tokens_tree_extra), " , Types = ",
len(list(np.unique(list_of_tree_tokenz_extra))), )
```

۴) حذف اعلام نگارشی

برای جداسازی حروف الفبایی و غیر الفبایی از RegexpTokenizer استفاده می‌شود. اگر ویژگی gaps آن را برابر با true قرار داده شود، کاراکترهای الفبایی و غیر الفبایی در مکان‌هایی که از هم فاصله داشته باشند جدا می‌شوند و اگر بین آن‌ها فاصله‌ای وجود نداشته باشد آن را باهم دسته‌بندی می‌کند؛ حال اگر آن را برابر با false بگذاریم، خروجی ما برابر با تمامی فاصله‌های موجود در متن خواهد بود و خود کاراکترهای الفبایی و غیر الفبایی نمایش داده نمی‌شوند. برای حذف علائم نگارشی و اعداد از متد معرفی شده در سوال می‌توان از \w+ یا (?!\d)\w+ استفاده کرد.



```
Question-1 Result (en):  Hello! Hope you're doing well. It is a sample short text.This is our 1 st assi
Question-1 Result (fa):  سلام! امیدوارم خوب باشید :-) . این یک متن کوتاه نمونه است.این تمرین ۱ ما است.
Question-2 Result:  hello! hope you're doing well. it is a sample short text.this is our 1 st assignmen
```

```
Question-4 Result (punctuation-removed of en): ['hello', 'hope', 'you', 're', 'doing', 'well', 'it', 'is', 'a', 'sample', 'short', 'text', 'this', 'is', 'our', '1', 'st', 'assignment']
Question-4 Result (punctuation-removed of fa): ['سلام', 'امیدوارم', 'خوب', 'باشید', 'این', 'یک', 'متن', 'کوتاه', 'نمونه', 'است', 'این', 'تمرین', '۱', 'ما', 'است']
Question-4 Result (punctuation-removed of beanstalk): ['once', 'upon', 'a', 'time', 'a', 'boy', 'named', 'jack', 'got', 'himself', 'into', 'the', 'biggest', 'most', 'humongous', 'heap', 'of', 'trouble', 'ever', 'it', 'all', 'started', 'when', 'jacks', 'mama', 'asked', 'him', 'to', 'milk', 'the', 'old', 'cow', 'but', 'jack', 'decided', 'he', 'was', 'tired', 'of', 'milking', 'cows', 'no', 'way', 'no', 'how', 'im', 'not', 'milking', 'this', 'brown', 'cow', 'now', 'said', 'jack', 'and', 'he', 'decided', 'to', 'sell', 'the', 'old', 'cow', 'so', 'hed', 'never', 'have', 'to', 'milk', 'it', 'again', 'jack', 'was', 'on', 'his', 'way', 'to', 'market', 'to', 'sell', 'the', 'cow', 'when', 'he', 'came', 'across', 'a', 'peddler', 'hi', 'mr', 'peddler', 'said', 'jack', 'where', 'are', 'you', 'headed', 'asked', 'the', 'peddler', 'im', 'going', 'to', 'sell', 'my', 'cow', 'at', 'the', 'market', 'jack', 'answered', 'why', 'sell', 'your', 'cow', 'asked', 'the', 'peddler', 'trade', 'her', 'for', 'beans', 'beans', 'asked', 'jack', 'not', 'just', 'any', 'kind', 'of', 'beans', 'said', 'the', 'peddler', 'magic', 'beans', 'what', 'do', 'they', 'do', 'asked', 'jack', 'they', 'do', 'magic', 'said', 'the', 'peddler', 'magic', 'sold', 'said', 'jack', 'and', 'he', 'traded', 'the', 'cow', 'for', 'three', 'magic', 'beans', 'jack', 'got', 'home', 'and', 'told', 'his', 'mama', 'he', 'had', 'sold', 'the', 'cow', 'so', 'he', 'wouldnt', 'have', 'to', 'milk', 'her', 'anymore', 'oh', 'dear', 'you', 'did', 'what', 'jacks', 'mama', 'asked', 'i', 'sold', 'her', 'for', 'magic', 'beans', 'said', 'jack', 'you', 'sold', 'a', 'cow', 'for', 'magic', 'beans', 'jacks', 'mama', 'couldnt', 'believe', 'what', 'jack', 'was', 'telling', 'her', 'theres', 'no', 'such', 'thing', 'as', 'magic', 'beans', 'she', 'said', 'as', 'she', 'threw', 'the', 'beans', 'out', 'the', 'window', 'w
```

کد نوشته شده برای این بخش در زیر ارائه شده است:

```python
# Question 4
regex_tokenizer = tokenize.RegexpTokenizer(r"\w+")

regexed_content_en = regex_tokenizer.tokenize(" ".join(list_of_tree_tokenz_en))
regexed_content_fa = regex_tokenizer.tokenize(" ".join(list_of_tree_tokenz_fa))
regexed_content_beanstalk = regex_tokenizer.tokenize(" ".join(list_of_tree_tokenz_beanstalk))
regexed_content_zahak = regex_tokenizer.tokenize(" ".join(list_of_tree_tokenz_zahak))
regexed_content_extra = regex_tokenizer.tokenize(" ".join(list_of_tree_tokenz_extra))
```

```
print ("Question-4 Result (punctuation-removed of en):", regexed_content_en)
print ("Question-4 Result (punctuation-removed of fa):", regexed_content_fa)
print ("Question-4 Result (punctuation-removed of beanstalk):",
regexed_content_beanstalk)
print ("Question-4 Result (punctuation-removed of zahak):",
regexed_content_zahak)
print ("Question-4 Result (punctuation-removed of extra):",
regexed_content_extra)
```

۵) مفهوم Stop Word

به مرحله تبدیل داده‌ها به چیزی قابل فهم برای رایانه پیش‌پردازش میگویند. یکی از فرم‌های اساسی پیش‌پردازش حذف کردن کلمات و داده‌های بلا استفاده است که در پردازش زبان طبیعی (NLTK) به این داده‌های بلا استفاده ایست واژه‌ها (stop words) گفته می‌شود. با این کار کامپیوتر می‌تواند داده‌های متنی را بهتر درک کند. در زبان انگلیسی می‌توان به حروف تعریف یا شمارشی مانند a، an، the، in، at و... که ماشین‌های جستجو آن‌ها را در نظر نمی‌گیرند اشاره کرد. در کتابخانه‌های پایتون فهرستی از این stop words ها برای ۱۶ زبان مختلف وجود دارد که از طریق nltk_data در دسترس است. تعداد تکرارها به شرح ذیل است:

ایست واژه‌ها	The	A	An	In	On
تکرار	۵۵	۸	۱	۳	۵

```
Question-1 Result (en):  Hello! Hope you're doing well. It is a sample short text.This is our 1 st assi
Question-1 Result (fa):  .تست ام ۱ ینرمت نیا.تست هنومن ه اتوک نتم کی نیا . (-: دیشاب بوخ م زاودیما !م الس
Question-2 Result:  hello! hope you're doing well. it is a sample short text.this is our 1 st assignmen
```

```
Question-4 Result (punctuation-removed of extra): ['when', 'one', 'door', 'of', 'happiness', 'closes',
'another', 'opens', 'but', 'often', 'we', 'look', 'so', 'long', 'at', 'the', 'closed', 'door', 'that',
'we', 'do', 'not', 'see', 'the', 'one', 'which', 'has', 'been', 'opened', 'for', 'us']
Question-5 Result (stop-words-removed of en): hello hope well sample short text 1 st assignment
```

کد نوشته شده برای این بخش در زیر ارائه شده است:

```
# Question 5
english_stop_words = set(stopwords.words("english"))

filtered_content_en = []
filtered_content_beanstalk = []
filtered_content_extra = []

for i in regexed_content_en:
    if i not in english_stop_words:
        filtered_content_en.append(i)

processed_content_en = " ".join(filtered_content_en)
```

```
for i in regexed_content_beanstalk:
    if i not in english_stop_words:
        filtered_content_beanstalk.append(i)

processed_content_beanstalk = " ".join(filtered_content_beanstalk)

for i in regexed_content_extra:
    if i not in english_stop_words:
        filtered_content_extra.append(i)

processed_content_extra = " ".join(filtered_content_extra)


print ("Question-5 Result (stop-words-removed of en):", processed_content_en)
```

۶) Stemming

برای ریشه‌یابی کلمات معمولاً از دو روش ریشه‌یابی (Stemming) و بُن‌واژه‌سازی (Lemmatization) استفاده می‌شود که هر دو روش در نهایت ریشه‌ی یک کلمه را به دست می‌آورند.

متد و الگوریتم‌های مختلفی جهت انجام عمل ریشه‌یابی وجود دارد که الگوریتم Porter و Lancaster's از الگوریتم‌های معروف در زبان انگلیسی برای استخراج فرم پایه است. این الگوریتم طبق یک سری قاعده‌ی منظم (مثلاً حذف حرف s در آخر کلماتِ جمع) می‌تواند ریشه‌ی کلمات را با دقتِ خوبی به دست آورد. خروجی متن نهایی پس از ریشه‌یابی لزوما کلمات با معنا و موجود در لغت‌نامه نخواهد بود اما پیشوندها و پسوندها حذف شده و در نهایت ساده‌ترین حالت کلمه و ریشه آن، به عنوان خروجی ارائه می‌شود.

```
Question-6 Result (PorterStemmer of en):
well  -->  well
Question-6 Result (LancasterStemmer of en):
well  -->  wel
Question-6 Result (PorterStemmer of beanstalk):
named  -->  name
started  -->  start
sell  -->  sell
jack  -->  jack
Question-6 Result (LancasterStemmer of beanstalk):
named  -->  nam
started  -->  start
sell  -->  sel
jack  -->  jack
```

کد نوشته شده برای این بخش در زیر ارائه شده است:

```
# Question 6

ps = stemmer.PorterStemmer()
lcs = stemmer.LancasterStemmer()

root_stemmer=[]
root_lancaster=[]

index_list=[2]
print ("Question-6 Result (PorterStemmer of en):")
for i in index_list:
    #root_stemmer.append(ps.stem(filtered_content_en[i]))
    print (filtered_content_en[i] , " --> " ,ps.stem(filtered_content_en[i]))

print ("Question-6 Result (LancasterStemmer of en):")
for i in index_list:
    #root_lancaster.append(lcs.stem(filtered_content_en[i]))
    print(filtered_content_en[i], " --> ", lcs.stem(filtered_content_en[i]))


index_list=[3,11,60,68]
print ("Question-6 Result (PorterStemmer of beanstalk):")
for i in index_list:
    #root_stemmer.append(ps.stem(filtered_content_beanstalk[i]))
    print (filtered_content_beanstalk[i] , " --> "
,ps.stem(filtered_content_beanstalk[i]))

print ("Question-6 Result (LancasterStemmer of beanstalk):")
for i in index_list:
    #root_lancaster.append(lcs.stem(filtered_content_beanstalk[i]))
    print(filtered_content_beanstalk[i], " --> ",
lcs.stem(filtered_content_beanstalk[i]))
```

۷) Lemmatization

Lemmatizer ابهام متن را به حداقل می‌رساند. کلمات مثالی مانند bicycle یا bicycles به کلمه پایه bicycle تبدیل می‌شوند. اساساً، تمام کلماتی را که معنی یکسان اما نمایش متفاوتی دارند به شکل پایه خود تبدیل می‌کند. تراکم کلمات را در متن پردازشی کاهش می‌دهد و به تهیه ویژگی‌های دقیق برای ماشین آموزشی کمک می‌کند. داده‌های تمیزتر، مدل یادگیری ماشین را هوشمندتر و دقیق‌تر خواهد کرد. NLTK Lemmatizer همچنین باعث صرفه جویی در حافظه و همچنین هزینه محاسباتی می‌شود.

حال آیا استفاده از متد Lemmatizer با ورودی‌های پیش‌فرض، برای همه این کلمات پاسخ درست را برمی‌گرداند؟ پاسخ، خیر است. این متد با ورودی پیش‌فرض روی همه کلمات به درستی کار نمی‌کند اما با تغییر ورودی پارامتر pos جواب به درستی محاسبه می‌شود. این پارامتر می‌تواند ورودی‌هایی مانند r، a، s و v داشته باشد که بیانگر فعل، صفت، قید یا اسم بودن آن کلمه می‌باشد. همچنین بهتر است از کلمات جمع استفاده نشود؛ زیرا es و s انتهای کلمات حذف می‌شوند و حتی شکل پایه کلمه گاهی اوقات بهم می‌رزد.

```
Question-7 Result (lemmatizer without pos):
{'word': 'went', 'type': 'v'}  -->  went
{'word': 'better', 'type': 'a'}  -->  better
{'word': 'was', 'type': 'v'}  -->  wa
{'word': 'eaten', 'type': 'v'}  -->  eaten
{'word': 'bufferfiles', 'type': 'n'}  -->  bufferfiles
{'word': 'fishing', 'type': 'n'}  -->  fishing
{'word': 'signaling', 'type': 's'}  -->  signaling
Question-7 Result (lemmatizer with pos):
{'word': 'went', 'type': 'v'}  -->  go
{'word': 'better', 'type': 'a'}  -->  good
{'word': 'was', 'type': 'v'}  -->  be
{'word': 'eaten', 'type': 'v'}  -->  eat
{'word': 'bufferfiles', 'type': 'n'}  -->  bufferfiles
{'word': 'fishing', 'type': 'n'}  -->  fishing
{'word': 'signaling', 'type': 's'}  -->  signaling
```

کد نوشته شده برای این بخش در زیر ارائه شده است:

```
# Question 7


"""
Lemmatize word data table
|    words    | types |
|+-----------+|+-----+|
|     went    |   v   |
|    better   |   a   |
|     was     |   v   |
|    eaten    |   v   |
| bufferfiles |   n   |
|   fishing   |   n   |
|  signaling  |   s   |
"""

# word type, type is in range (v: verb | n: nouns | r: adverbs | a: adjective
| s: satelliteAdjective)
```

```
list_of_words = [
    {
        "word": "went",
        "type": "v",
    },
    {
        "word": "better",
        "type": "a",
    },
    {
        "word": "was",
        "type": "v",
    },
    {
        "word": "eaten",
        "type": "v",
    },
    {
        "word": "bufferfiles",
        "type": "n",
    },
    {
        "word": "fishing",
        "type": "n",
    },
    {
        "word": "signaling",
        "type": "s",
    },
]


lemmatizer = WordNetLemmatizer()

root_lemmatizer = []

print ("Question-7 Result (lemmatizer without pos):")
for i in list_of_words:
    print(i, " --> ", lemmatizer.lemmatize(i["word"]))

print ("Question-7 Result (lemmatizer with pos):")
for i in list_of_words:
        print(i, " --> ", lemmatizer.lemmatize(i["word"], i["type"]))
```