WordNet

WordNet is a lexical database that of nouns, verbs, adverbs, and adjectives and organizes those words into sets of synonyms known as synsets. WordNet also organizes words in a hierarchal structure. Synsets contain short definitions called glosses, examples, and relation to other words.

Libraries imported needed for this assignment

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('book')
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
from math import log2
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to /root/nltk_data...
[nltk_data]    |   Package ieer is already up-to-date!
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package nps_chat to /root/nltk_data...
[nltk_data]    |   Package nps_chat is already up-to-date!
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package ppattach to /root/nltk_data...
[nltk_data]    |   Package ppattach is already up-to-date!
[nltk_data]    | Downloading package reuters to /root/nltk_data...
[nltk_data]    |   Package reuters is already up-to-date!
[nltk_data]    | Downloading package senseval to /root/nltk_data...
[nltk_data]    |   Package senseval is already up-to-date!
[nltk_data]    | Downloading package state_union to /root/nltk_data...
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package swadesh to /root/nltk_data...
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package timit to /root/nltk_data...
[nltk_data]    |   Package timit is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data...
```

```
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package toolbox to /root/nltk_data...
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package udhr to /root/nltk_data...
[nltk_data]    |   Package udhr is already up-to-date!
```

(3)

```
synsets = wn.synsets('car')
print('Synsets of car: ', synsets)

synset = synsets[0]

print('Selected synset:', )
print('Definition:', synset.definition())
print('Usage examples:', synset.examples())
print('Lemmas:', synset.lemmas())

#Traverse the hierarchy
top = wn.synset('entity.n.01')
while synset:
    print(synset)
    if synset == top:
        break
    if synset.hypernyms():
        synset = synset.hypernyms()[0]
```

```
Synsets of car:  [Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'), Synset('cable_car.n.01')]
Selected synset:
Definition: a motor vehicle with four wheels; usually propelled by an internal combustion engine
Usage examples: ['he needs a car to get to work']
Lemmas: [Lemma('car.n.01.car'), Lemma('car.n.01.auto'), Lemma('car.n.01.automobile'), Lemma('car.n.01.machine'), Lemma('car.
Synset('car.n.01')
Synset('motor_vehicle.n.01')
Synset('self-propelled_vehicle.n.01')
Synset('wheeled_vehicle.n.01')
Synset('container.n.01')
Synset('instrumentality.n.03')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

I noticed that for the organization of nouns, the higher in the hierarchy that it gets for a word, it becomes more generalized. To describe a car or any other physical object in the world, we're able to refer to those as entities, which is the highest synset in the hierarchy when it comes to nouns.

Output Hyponyms,

```
print("Hypernyms:", synset.hypernyms())
print("Hyponyms:", synset.hyponyms())
print("Meronyms:", synset.part_meronyms())
print("Holonyms:", synset.part_holonyms())
print("Antonyms:", synset.lemmas()[0].antonyms())
```

```
Hypernyms: []
Hyponyms: [Synset('abstraction.n.06'), Synset('physical_entity.n.01'), Synset('thing.n.08')]
Meronyms: []
Holonyms: []
Antonyms: []
```

Output synsets of verb (Run)

```
run = wn.synsets('run', pos='v')

print('Synsets of run: ', run)

#select a synset
synset_run = run[0]

print("Synset:", synset_run)
print("Definition:", synset_run.definition())
print("Examples:", synset_run.examples())
```

```
print("Lemmas:", synset_run.lemmas())

top = wn.synset('travel.v.01')
while synset_run:
    print(synset_run)
    if synset_run == top:
      break
    if synset_run.hypernyms():
        synset_run = synset_run.hypernyms()[0]
```

```
Synsets of run:  [Synset('run.v.01'), Synset('scat.v.01'), Synset('run.v.03'), Synset('operate.v.01'), Synset('run.v.05'), S
Synset: Synset('run.v.01')
Definition: move fast by using one's feet, with one foot off the ground at any given time
Examples: ["Don't run--you'll be out of breath", 'The children ran to the store']
Lemmas: [Lemma('run.v.01.run')]
Synset('run.v.01')
Synset('travel_rapidly.v.01')
Synset('travel.v.01')
```

The traversal for my word, run, a verb contains a list much smaller than the hierarchal list for nouns. Similar to the traversal for a noun, it contains a more general synset as the word travel can have a bunch of meanings and similar words such as walking or flying.

Use Morphy to find different forms of the word (run)

```
print(wn.morphy('run', wn.ADJ))
print(wn.morphy('run', wn.VERB))
print(wn.morphy('run', wn.NOUN))
```

```
None
run
run
```

```
robot = wn.synset('robot.n.01')
cyborg = wn.synset('machine.n.01')


#Wu Palmer similarity metric
print(wn.wup_similarity(robot, cyborg))

#Lesk Algorithm

sent = ['Are', 'you', 'human', 'or', 'a', 'robot']
print(lesk(sent, 'robot', 'n'))
```

```
0.8235294117647058
Synset('automaton.n.02')
```

I used the words robot and machine to compare using the Wu Palmer metric which gave a result of 0.8. I find that result to be fairly accurate since I consider the two words to be similar in the sense of automation. The lesk algorithm outputted a result I was sort of looking for in the context of how captchas are used to make sure that an actual human is attempting to access the website and not an automated bot/program.

SentiWordNet is a library that is used to analyze the attitude of text and outputting a score of positivity, negativity and objectivity. Potential use cases could be analyzing tweets during a disastrous time such as Covid-19 pandemic and scraping tweets to see if the things are getting better or worse in the world.

```
emotion_word= 'grief'

# Find senti-synsets

grief_sentisets = list(swn.senti_synsets(emotion_word))

for synset in grief_sentisets:
  print(synset)
  print("Pos:", synset.pos_score(), 'Neg:', synset.neg_score(), 'Obj:', synset.obj_score())
```

```
<grief.n.01: PosScore=0.0 NegScore=0.625>
Pos: 0.0 Neg: 0.625 Obj: 0.375
<grief.n.02: PosScore=0.375 NegScore=0.25>
Pos: 0.375 Neg: 0.25 Obj: 0.375
```

Make up a sentence. Output the polarity for each word in the sentence

```
sent = 'The boy had the best day of his life because he aced his exam'
tokens = sent.split()

for token in tokens:
  sentisyns = list(swn.senti_synsets(token))
  if sentisyns:
    print(token, ':', 'Pos:', sentisyns[0].pos_score(), 'Neg:', sentisyns[0].neg_score(), 'Obj:', sentisyns[0].obj_score())
  else:
    print(token ,': No score')
```

```
The : No score
boy : Pos: 0.25 Neg: 0.0 Obj: 0.75
had : Pos: 0.25 Neg: 0.0 Obj: 0.75
the : No score
best : Pos: 0.25 Neg: 0.0 Obj: 0.75
day : Pos: 0.0 Neg: 0.0 Obj: 1.0
of : No score
his : No score
life : Pos: 0.0 Neg: 0.0 Obj: 1.0
because : No score
he : Pos: 0.0 Neg: 0.0 Obj: 1.0
aced : Pos: 0.5 Neg: 0.0 Obj: 0.5
his : No score
exam : Pos: 0.0 Neg: 0.0 Obj: 1.0
```

The sentence I provided was a very positive statement and some of the scores that were outputted don't reflect well on the sentiment that was intended. For example, I expected best to have a higher posivity score than 0.25 and aced was a 0.5.

Collocations are words that are grouped together (2 or more) and are unable to be substituted for other words. Certain words in collocations cannot be substituted for other synonyms because it may change the meaning that the collocation is trying to convey.

```
text4.collocations()

# collocation: foreign nations
text = ' '.join(text4.tokens)
length = len(set(text))


x = text.count('fellow') / length
y = text.count('citizens') / length
xy = text.count('fellow citizens') / length

pmi = log2(xy / (x * y))
print('PMI Score:', pmi)
```

```
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
PMI Score: -2.851792919669711
```

A negative PMI indicates that the collocation (x+y) isn't actually a collocatiion. This PMI score result makes sense because it isn't really a collocation since the two words can be substituted to convey the same meaning.

✓ 0s completed at 4:07 PM