Name: Abhejay Murali

Assignment: WordNet

WordNet is a project that was started by Princeton University during the 1980s by Psychologist George Miller who was interested in seeing how people organize different concepts. It contains a hierarchical organization of nouns, verbs, adjectives, and adverbs. Some of the things would be glosses (short definitions), synsets (synonym sets), use case examples, and relationship to other words.

```
import nltk
nltk.download('popular')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
from nltk.corpus import wordnet as wn
```

```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gazetteers.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/shakespeare.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/stopwords.zip.
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/twitter_samples.zip.
[nltk_data]    | Downloading package omw to /root/nltk_data...
[nltk_data]    | Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    | Downloading package wordnet2021 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet31 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/wordnet_ic.zip.
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/words.zip.
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Unzipping tokenizers/punkt.zip.
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection popular
[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]   Unzipping corpora/sentiwordnet.zip.
```

STEP 2

Select a noun. Output all synsets.

```
# get all synsets of 'exercise'
wn.synsets('answer')
```

```
[Synset('answer.n.01'),
 Synset('solution.n.02'),
 Synset('answer.n.03'),
 Synset('answer.n.04'),
 Synset('answer.n.05'),
 Synset('answer.v.01'),
 Synset('answer.v.02'),
 Synset('answer.v.03'),
 Synset('answer.v.04'),
 Synset('answer.v.05'),
 Synset('answer.v.06'),
 Synset('suffice.v.01'),
 Synset('answer.v.08'),
 Synset('answer.v.09'),
 Synset('answer.v.10')]
```

STEP 3

1) Select one synset from the list of synsets.

2) Extract its definition, usage examples, and lemmas.

3) From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

EXPLANATION: Nouns are pretty simple. You can gather the synset and use wordnet to extract valuable information such as the definition, different examples, and lemmas. Something specific to nouns is that there is a top level synset which we use in order to traverse through the hierarchy. When traversing, you can see the word entity being used as the top level synset.

```
# get a definition for the first noun synset
wn.synset('answer.n.01').definition()
```

```
'a statement (either spoken or written) that is made to reply to a question or request or c
riticism or accusation'
```

```
# extract examples
wn.synset('answer.n.01').examples()
```

```
['I waited several days for his answer',
 'he wrote replies to several of his critics']
```

```
# extract lemmas
wn.synset('answer.n.01').lemmas()
```

```
[Lemma('answer.n.01.answer'),
 Lemma('answer.n.01.reply'),
 Lemma('answer.n.01.response')]
```

```
# Traversing word hierarchy
answer = wn.synset('answer.n.01')
ans = answer.hypernyms()[0]
top = wn.synset('entity.n.01')
while ans:
  print(ans)
  if ans == top:
    break
  if ans.hypernyms():
    ans = ans.hypernyms()[0]
```

```
Synset('statement.n.01')
Synset('message.n.02')
Synset('communication.n.02')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

STEP 4

Output the following (or an empty list if none exist): hypernyms, hyponyms, meronyms, holonyms, antonym.

```
from nltk.corpus.reader import wordnet
# STEP 4
print('hypernyms:', answer.hypernyms())
print('hyponyms:', answer.hyponyms())
```

```
print('meronyms:', answer.part_meronyms())
print('holonyms:', answer.part_holonyms())

antonyms = []
for synset in wn.synsets('answer'):
  for lemma in synset.lemmas():
    if lemma.antonyms():
      antonyms.append(lemma.antonyms()[0].name())

print('Antonyms:', str(antonyms))
```

```
     hypernyms: [Synset('statement.n.01')]
     hyponyms: [Synset('feedback.n.02'), Synset('rescript.n.01')]
     meronyms: []
     holonyms: []
     Antonyms: ['question']
```

STEP 5

Select a verb. Output all synsets.

```
# STEP 5
# get all synsets of 'exercise'
wn.synsets('think')
```

```
     [Synset('think.n.01'),
      Synset('think.v.01'),
      Synset('think.v.02'),
      Synset('think.v.03'),
      Synset('remember.v.01'),
      Synset('think.v.05'),
      Synset('think.v.06'),
      Synset('intend.v.01'),
      Synset('think.v.08'),
      Synset('think.v.09'),
      Synset('think.v.10'),
      Synset('think.v.11'),
      Synset('think.v.12'),
      Synset('think.v.13')]
```

STEP 6

1) Select one synset from the list of synsets.

2) Extract its definition, usage examples, and lemmas.

3) From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

EXPLANATION: Verbs are also pretty simple just like nouns. You can gather the synset and use wordnet to extract valuable information such as the definition, different examples, and lemmas. Something specific to verbs is that there is no top level synset for all verbs. When traversing through the word hierarchy, you can see that a lambda function is used.

```
# get a definition for the first noun synset
wn.synset('think.v.01').definition()
```

```
     'judge or regard; look upon; judge'
```

```
# extract examples
wn.synset('think.v.01').examples()
```

```
     ['I think he is very smart',
      'I believe her to be very smart',
      'I think that he is her boyfriend',
      'The racist conceives such people to be inferior']
```

```
# extract lemmas
wn.synset('think.v.01').lemmas()
```

```
     [Lemma('think.v.01.think'),
      Lemma('think.v.01.believe'),
      Lemma('think.v.01.consider'),
      Lemma('think.v.01.conceive')]
```

```
# Traversing word hierarchy
hyper = lambda s: s.hypernyms()
```

```
think = wn.synset('think.v.01')

list(think.closure(hyper))
```

```
[Synset('evaluate.v.02'), Synset('think.v.03')]
```

▼ STEP 7

Use morphy to find as many different forms of the word as you can.

```
print(wn.morphy('think'))
print(wn.morphy('think', wn.VERB))
```

```
think
think
```

▼ STEP 8

1) Select two words that you think might be similar.

2) Find the specific synsets you are interested in.

3) Run the Wu-Palmer similarity metric and the Lesk algorithm.

EXPLANATION: There are many ways to check word similarity. As you can see the two words selected are 'monkey' and 'gorilla'. They are borth similar animals. The Wu-Palmer algorithm is used to calculate the similarity between the two words and is outputted by a decimal which essentially gave around 82-83%. The second situation to compare word similarity is used in the Lesk algorithm. Based on the usage of a word in a sentence, the Lesk algorithm is able to show which definition from the synset is the word and it's context is closest in use.

```
monkey = wn.synset('monkey.n.01')
gorilla = wn.synset('gorilla.n.01')

wn.wup_similarity(monkey, gorilla)
```

```
0.8275862068965517
```

```
from nltk.wsd import lesk

for ss in wn.synsets('monkey'):
  print(ss, ss.definition())
```

```
Synset('monkey.n.01') any of various long-tailed primates (excluding the prosimians)
Synset('imp.n.02') one who is playfully mischievous
Synset('tamper.v.01') play around with or alter or falsify, usually secretively or dishonestly
Synset('putter.v.02') do random, unplanned work or activities or spend time idly
```

```
zoo = ['I', 'went', 'to', 'the', 'zoo', 'and', 'saw', 'a', 'monkey', '.']
print(lesk(zoo, 'monkey', 'n'))
print(lesk(zoo, 'monkey'))
```

```
Synset('monkey.n.01')
Synset('monkey.n.01')
```

▼ STEP 9

1) Select an emotionally charged word.

2) Find its senti-synsets and output the polarity scores for each word.

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

SentiWordNet: This is a tool that is used for Sentiment Analysis from NLTK. It is a lexical resource used for opinion mining. It assigns each synset in wordnet three different scores (positivity, negativity, objectivity). The tool can be used to identify what context this word would possibly be used in and emotion that is being shown off the word alone.

EXPLANATION: The word that I chose to use is lonely. First WordNet is utilized in order to output the synsets and each of these synset is assigned a positive and negative score. The same function is also used in a sentence but it also showing the objective score. These polarity scores are used to show the connotation or emotion that is being shown in this word. It is achieving this by assigning a score that adds up to a total of 1 and how much of it sounds positive, negative, or objective. Knowing these scores could help in a variety of situations in NLP. The first one that comes into mind is understanding more of what the user is trying to communicate by seeing the emotion that is most likely being described by specific words.

```
from nltk.corpus import sentiwordnet as swn


senti_list = list(swn.senti_synsets('lonely'))
for item in senti_list:
  print(item)
```

```
    <alone.s.02: PosScore=0.125 NegScore=0.25>
    <lonely.s.02: PosScore=0.25 NegScore=0.5>
    <lone.s.02: PosScore=0.125 NegScore=0.0>
    <lonely.s.04: PosScore=0.0 NegScore=0.0>
```

```
sentence = 'The exam was very hard'

tokens = sentence.split()

for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        print(token)
        print("negative: ", syn.neg_score())
        print("positive: ", syn.pos_score())
        print("objective: ", syn.obj_score())
```

```
    exam
    negative:  0.0
    positive:  0.0
    objective:  1.0
    was
    negative:  0.0
    positive:  0.0
    objective:  1.0
    very
    negative:  0.0
    positive:  0.5
    objective:  0.5
    hard
    negative:  0.75
    positive:  0.0
    objective:  0.25
```

STEP 10

1) Output collocations for text4, the Inaugural corpus.

2) Select one of the collocations identified by NLTK.

3) Calculate mutual information.

Collocation: This is when two or more words combine more often than expected and also cannot be substituted by a word and get the same meaning. Collocations can be identified by pmi (point-wise mutual information).

EXPLANATION: First in this step, nltk text is being imported. Once this is done, the collocations are identified by a simple command, which shows many different collocations. Then the math package is imported to calculate the mutual information. The overall length of the text from NLTK is put into a set to keep only the unique vocab. Then a select collocation is used in which each word in the collocation is divided by the overall vocab. Then we use the log function to calculate the pmi. PMI of 0 means x and y are independent. PMI that is positive, then likely to be a collocation. PMI that is negative, not likely to be a collocation

```
nltk.download('webtext')
from nltk.book import *
```

```
    [nltk_data] Downloading package webtext to /root/nltk_data...
    [nltk_data]   Unzipping corpora/webtext.zip.
    *** Introductory Examples for the NLTK Book ***
    Loading text1, ..., text9 and sent1, ..., sent9
    Type the name of the text or sentence to view it.
    Type: 'texts()' or 'sents()' to list the materials.
    text1: Moby Dick by Herman Melville 1851
    text2: Sense and Sensibility by Jane Austen 1811
    text3: The Book of Genesis
    text4: Inaugural Address Corpus
    text5: Chat Corpus
    text6: Monty Python and the Holy Grail
    text7: Wall Street Journal
    text8: Personals Corpus
    text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

```
text4.collocations()
```

```
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
```

```
import math
vocab = len(set(text4))
fc = text.count('fellow citizens')/vocab
print('p(fellow citizens) =', fc)
f = text.count('fellow')/vocab
print('p(fellow) =', f)
c = text.count('citizens')/vocab
print('p(citizens) =', f)
pmi = math.log2(fc/(f*c))
print('pmi =', pmi)
```

```
p(fellow citizens) = 0.006084788029925187
p(fellow) = 0.013665835411471322
p(citizens) = 0.013665835411471322
pmi = 4.0472042737811735
```

✓ 0s completed at 9:30 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.