In [ ]:
```
from nltk.corpus import wordnet as wn
```

WordNet is a database of Nouns, Verbs, Adjectives,and Adverbs that is organized in a hierachy. Each entry includes a short definition, synonyms, use examples, and relations to other words.

In [ ]:
```
# Pick a synsets
memory = wn.synsets('memory')

#  Output all the synsets
memory_synsets = wn.synsets('memory', pos=wn.NOUN)
for sense in memory_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + str(lemmas))
```

```
Synset: memory.n.01(something that is remembered)
         Lemmas:['memory']
Synset: memory.n.02(the cognitive processes whereby past experience is remembered)
         Lemmas:['memory', 'remembering']
Synset: memory.n.03(the power of retaining and recalling past experience)
         Lemmas:['memory', 'retention', 'retentiveness', 'retentivity']
Synset: memory.n.04(an electronic memory device)
         Lemmas:['memory', 'computer_memory', 'storage', 'computer_storage', 'store', 'memory_board']
Synset: memory.n.05(the area of cognitive psychology that studies memory processes)
         Lemmas:['memory']
```

In [ ]:
```
memory = wn.synset('memory.n.04')

print(memory.definition())
print(memory.examples())
print(memory.lemmas())

# WordNet hierarchy
hyp = memory.hypernyms()[0]
top = memory.root_hypernyms()[0]
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
an electronic memory device
['a memory and the CPU form the central part of a computer to which peripherals are attached']
[Lemma('memory.n.04.memory'), Lemma('memory.n.04.computer_memory'), Lemma('memory.n.04.storage'), Lemma('memory.n.04.computer_
storage'), Lemma('memory.n.04.store'), Lemma('memory.n.04.memory_board')]
Synset('hardware.n.03')
Synset('component.n.03')
Synset('part.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Nouns are organized in a hierachy based on the hypernymy/hyponymy relation between synsets. Nouns are all highly-connected with one another and connect to the top-most level labeled 'entity'.

In [ ]:
```
print(memory.hypernyms())
print(memory.hyponyms())
print(memory.part_meronyms())
print(memory.part_holonyms())

# Loop over the lemmas and check for antonyms
for i in range(len(memory.lemmas())):
   print(memory.lemmas()[i].antonyms())
```

```
[Synset('hardware.n.03'), Synset('memory_device.n.01')]
[Synset('non-volatile_storage.n.01'), Synset('read-only_memory.n.01'), Synset('real_storage.n.01'), Synset('scratchpad.n.01'),
Synset('virtual_memory.n.01'), Synset('volatile_storage.n.01')]
[Synset('register.n.04')]
[Synset('computer.n.01')]
[]
[]
[]
[]
[]
[]
```

In [ ]:
```
# Pick a synsets
visit = wn.synsets('visit')

#  Output all the synsets
visit_synsets = wn.synsets('visit', pos=wn.VERB)
for sense in visit_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + str(lemmas))
```

```
Synset: visit.v.01(go to see a place, as for entertainment)
         Lemmas:['visit', 'see']
Synset: travel_to.v.01(go to certain places as for sightseeing)
         Lemmas:['travel_to', 'visit']
Synset: visit.v.03(pay a brief visit)
         Lemmas:['visit', 'call_in', 'call']
Synset: visit.v.04(come to see in an official or professional capacity)
         Lemmas:['visit', 'inspect']
Synset: inflict.v.01(impose something unpleasant)
         Lemmas:['inflict', 'bring_down', 'visit', 'impose']
Synset: chew_the_fat.v.01(talk socially without exchanging too much information)
         Lemmas:['chew_the_fat', 'shoot_the_breeze', 'chat', 'confabulate', 'confab', 'chitchat', 'chit-chat', 'chatter', 'cha
ffer', 'natter', 'gossip', 'jaw', 'claver', 'visit']
Synset: visit.v.07(stay with as a guest)
         Lemmas:['visit']
Synset: visit.v.08(assail)
         Lemmas:['visit']
```

In [ ]:
```
visit = wn.synset('visit.v.01')

print(visit.definition())
print(visit.examples())
print(visit.lemmas())


# WordNet hierarchy
hyp = visit.hypernyms()[0]
top = visit.root_hypernyms()[0]
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
go to see a place, as for entertainment
['We went to see the Eiffel Tower in the morning']
[Lemma('visit.v.01.visit'), Lemma('visit.v.01.see')]
Synset('tour.v.01')
Synset('travel.v.02')
Synset('travel.v.03')
```

Verbs are similar to nouns in that they are organized in a hierachy based on the hypernymy/hyponymy relation between synsets. However, verbs do not have a common top-level synset to reference to.

In [ ]:
```
print(wn.morphy('visit'))
print(wn.morphy('visit', wn.NOUN))
print(wn.morphy('visit', wn.ADJ))
print(wn.morphy('visit', wn.ADV))
print(wn.morphy('visit', wn.VERB))
```

```
visit
visit
None
None
visit
```

In [ ]:
```
baseball_synsets = wn.synsets('baseball', pos=wn.NOUN)
for sense in baseball_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + str(lemmas))

football_synsets = wn.synsets('football', pos=wn.NOUN)
for sense in football_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + str(lemmas))
```

```
Synset: baseball.n.01(a ball game played with a bat and ball between two teams of nine players; teams take turns at bat trying
to score runs)
        Lemmas:['baseball', 'baseball_game']
Synset: baseball.n.02(a ball used in playing baseball)
        Lemmas:['baseball']
Synset: football.n.01(any of various games played with a ball (round or oval) in which two teams try to kick or carry or prope
l the ball into each other's goal)
        Lemmas:['football', 'football_game']
Synset: football.n.02(the inflated oblong ball used in playing American football)
        Lemmas:['football']
```

In [ ]:
```
from nltk.wsd import lesk
```

```
Synset('bank.n.01') sloping land (especially the slope beside a body of water)
Synset('depository_financial_institution.n.01') a financial institution that accepts deposits and channels the money into lend
ing activities
Synset('bank.n.03') a long ridge or pile
Synset('bank.n.04') an arrangement of similar objects in a row or in tiers
Synset('bank.n.05') a supply or stock held in reserve for future use (especially in emergencies)
Synset('bank.n.06') the funds held by a gambling house or the dealer in some gambling games
Synset('bank.n.07') a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effec
ts of centrifugal force
Synset('savings_bank.n.02') a container (usually with a slot in the top) for keeping money at home
Synset('bank.n.09') a building in which the business of banking transacted
Synset('bank.n.10') a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
Synset('bank.v.01') tip laterally
Synset('bank.v.02') enclose with a bank
Synset('bank.v.03') do business with a bank or keep an account at a bank
Synset('bank.v.04') act as the banker in a game or in gambling
Synset('bank.v.05') be in the banking business
Synset('deposit.v.02') put into a bank account
Synset('bank.v.07') cover with ashes so to control the rate of burning
Synset('trust.v.01') have confidence or faith in
```

In [ ]:
```
# Wu-Palmer

baseball = wn.synset('baseball.n.01')
football = wn.synset('football.n.01')

print(wn.wup_similarity(baseball, football))
```

```
0.88
Synset('baseball.n.01') a ball game played with a bat and ball between two teams of nine players; teams take turns at bat tryi
ng to score runs
Synset('baseball.n.02') a ball used in playing baseball
```

In [ ]:
```
# Lesk algorithm

from nltk.wsd import lesk

# look at the definitions for 'football'
for ss in wn.synsets('football'):
    print(ss, ss.definition())


sentence = 'The quarterback threw the football for a touchdown.'
print(lesk(sentence, 'football'))
```

```
Synset('football.n.01') any of various games played with a ball (round or oval) in which two teams try to kick or carry or pro
pel the ball into each other's goal
Synset('football.n.02') the inflated oblong ball used in playing American football
Synset('football.n.01')
```

The results for the two algorithms were very different. The Wu-Palmer showed that the two words were strongly related to each other. This makes sense as both words are commonly played sports. The lesk algorithm was not entirely accurate on the definitions. The definition I used in the sentence was refering to the ball whereas the algorithm returned the definition for the game.

SentiWordNet allows for a computer to interpret the tone of a particular sentence. It works by assining 3 scores to a word that rates how positive, negative, or objective it is. To get the analysis for an entire sentence, you would have to add up the positive/negative counts for each word and then you have the overall analysis.

In [ ]:
```
import nltk
from nltk.corpus import sentiwordnet as swn
```

In [ ]:
```
fast_synsets = wn.synsets('starving')
for sense in fast_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + str(lemmas))
```

```
Synset: starvation.n.02(the act of depriving of food or subjecting to famine)
         Lemmas:['starvation', 'starving']
Synset: starve.v.01(be hungry; go without food)
         Lemmas:['starve', 'hunger', 'famish']
Synset: starve.v.02(die of food deprivation)
         Lemmas:['starve', 'famish']
Synset: starve.v.03(deprive of food)
         Lemmas:['starve', 'famish']
Synset: crave.v.01(have a craving, appetite, or great desire for)
         Lemmas:['crave', 'hunger', 'thirst', 'starve', 'lust']
Synset: starve.v.05(deprive of a necessity and cause suffering)
         Lemmas:['starve']
Synset: starved.s.01(suffering from lack of food)
         Lemmas:['starved', 'starving']
```

In [ ]:
```
senti_list = list(swn.senti_synsets('starve'))
for item in senti_list:
    print(item)
```

```
<starve.v.01: PosScore=0.0 NegScore=0.25>
<starve.v.02: PosScore=0.0 NegScore=0.0>
<starve.v.03: PosScore=0.0 NegScore=0.0>
<crave.v.01: PosScore=0.5 NegScore=0.0>
<starve.v.05: PosScore=0.0 NegScore=0.5>
```

In [ ]:
```
sentence = 'there was way too much homework assigned over the weekend'
tokens = sentence.split()
for token in tokens:
   syn_list = list(swn.senti_synsets(token))
   if syn_list:
      syn = syn_list[0]
      pos = syn.pos_score()
      neg = syn.neg_score()

      print(f'{token:15}:  Positive:  {pos:5}    Negative:  {neg:5}')
```

```
there          :  Positive:    0.0    Negative:    0.0
was            :  Positive:    0.0    Negative:    0.0
way            :  Positive:    0.0    Negative:    0.0
too            :  Positive:  0.125    Negative:   0.25
much           :  Positive:  0.125    Negative:  0.125
homework       :  Positive:    0.0    Negative:    0.0
assigned       :  Positive:    0.0    Negative:    0.0
over           :  Positive:    0.0    Negative:    0.0
weekend        :  Positive:    0.0    Negative:    0.0
```

The overall scores were not as negative as I feel that they should be. I wrote the sentence trying to get a negative result and the output only slightly leans negative. I think the issue is the word 'way' has a score of 0 whereas the phrase 'way too much' implies a more negative tone than 'too much.'

A collocation is predictable combination of words that are put together for a specific meaning. The words can be anything from nouns, verbs, adverbs, and adjectives. The words in the collocation can not be subsitited while keeping the same meaning. For example, 'heavy rain' is a no the same as 'strong rain.'

In [ ]:
```
from nltk.book import text4

print(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
None
```

In [ ]:
```
import math
text = ' '.join(text4.tokens)
vocab = len(set(text4))

hg = text.count('United States')/vocab
print("p(United States) = ",hg )
h = text.count('United')/vocab
print("p(United) = ", h)
g = text.count('States')/vocab
print('p(States) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(United States) =  0.015860349127182045
p(United) =  0.0170573566084788
p(States) =  0.03301745635910224
pmi =  4.815657649820885
```

United States is very much so a collocation. It represents the name of a country as a whole.