Wordnet is an organized database of english words that are arranged in a hierarchical structure according to their meaning. It includes the major parts of speech- nouns, verbs, adjectives, and adverbs- with their definitions and relationships to other words. The words are organized into sets of synonymous words, or synsets. The synsets are related in a tree structure, with the more specific a word is, the further it is down the tree. It has many functions that can be used to identify and understand the various relationships between words.

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Noun car synsets:

```
print(wn.synsets('car'))
```

```
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'), Synset(
```

Car synset 01 definition, examples, and lemmas:

```
s1= wn.synsets('car')[0]
print(s1)
print("\ncar.n.01 definition:")
print(s1.definition())
print("\ncar.n.01 examples:")
print(s1.examples())
print("\ncar.n.01 lemmas:")
print(s1.lemmas())
```

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

car.n.01 definition:
a motor vehicle with four wheels; usually propelled by an internal combustion engine

car.n.01 examples:
['he needs a car to get to work']

car.n.01 lemmas:
[Lemma('car.n.01.car'), Lemma('car.n.01.auto'), Lemma('car.n.01.automobile'), Lemma('car
```

Car synset 01 hypernyms up to top:

```
s1= wn.synsets('car')[0]
print("car.n.01 hypernyms up to top:")
hyp = s1.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
  print(hyp)
  if hyp == top:
    break
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
```

```
car.n.01 hypernyms up to top:
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')
```

Nouns in wordnet are organized as a single large heirarchy, with the top being the most general word: entity. Each level down has many words that are related to the parent word but more specific.

Car synset 01 hypernyms, hyponyms, meronyms, holonyms, antonym:

```
s1= wn.synsets('car')[0]
print("\ncar.n.01 hypernyms:")
print(s1.hypernyms())
print("\ncar.n.01 hyponyms:")
print(s1.hyponyms())
print("\ncar.n.01 meronyms:")
print(s1.part_meronyms())
print("\ncar.n.01 holonyms:")
print(s1.part_holonyms())
print("\ncar.n.01 antonym:")
print(s1.lemmas()[0].antonyms())
```

```
car.n.01 hypernyms:
[Synset('motor_vehicle.n.01')]
```

```
car.n.01 hyponyms:
[Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bus.n.04'), Synset('cab.n

car.n.01 meronyms:
[Synset('accelerator.n.01'), Synset('air_bag.n.01'), Synset('auto_accessory.n.01'), Syns

car.n.01 holonyms:
[]

car.n.01 antonym:
[]
```

Verb consume synsets:

```
print(wn.synsets('eat'))
```

```
[Synset('eat.v.01'), Synset('eat.v.02'), Synset('feed.v.06'), Synset('eat.v.04'), Synset
```

Consume synset definition, examples, and lemmas:

```
s1= wn.synsets('eat')[0]
print(s1)
print("\ndefinition:")
print(s1.definition())
print("\nexamples:")
print(s1.examples())
print("\nlemmas:")
print(s1.lemmas())
```

```
Synset('eat.v.01')

definition:
take in solid food

examples:
['She was eating a banana', 'What did you eat for dinner last night?']

lemmas:
[Lemma('eat.v.01.eat')]
```

Consume synset hypernyms up to top:

```
s1= wn.synsets('eat')[0]
print("hypernyms up to top:")
hyp = s1.hypernyms()[0]
```

```
while hyp:
  print(hyp)
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
  else:
    break
```

```
 hypernyms up to top:
 Synset('consume.v.02')
```

Wordnet synsets for verbs are organized in a less centralized way compared to nouns. There are heirarchical relationships, but these are not necessary, and all the words are not connected. Synsets are only related to words that are closely related and may not have many relations at all.

All forms of word using morphy:

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

```
 eat
 None
 eat
 None
```

Wu Palmer similarity is used to find how similar two words are in meaning by comparing how similar their paths are in the wordnet hierarchy from the top to the words. It is not a perfect method but does offer good results, often better than the basic path similarity algorithm. When given the words coffee and tea, it correctly identified how close the words were with a result of 0.888 repeating.

```
coffeeSet = wn.synsets('coffee')[0]
teaSet = wn.synsets('tea')[0]
print(wn.wup_similarity(coffeeSet, teaSet))
```

```
 0.8888888888888888
```

The Lesk Algorithm is used for word sense disambiguation, or understanding the specifc meaning of a word with multuple definitions based on the context. After using it on the sentence I provided, it actually gave me an incorrect result, providing the wrong definition for the context. This shows it isn't foolproof.

```
from nltk import word_tokenize
nltk.download('punkt')
from nltk.wsd import lesk
sent = ("I need to go back and do the next set of problems")
tokens = word_tokenize(sent)
setlesk=(lesk(tokens, 'set', 'n'))
print(setlesk)
```

```
Synset('stage_set.n.01')
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

SentiWordNet is built on WordNet and enables sentiment analysis of specific words, assigning them a positive, negative, and objective score based on their connotations. It is useful as it provides more that simply a word's definition and enables more understanding, going past simple word processing. It would help understand the tone of text, which could be used to figure out what type of text is being processed or what tone should be used to respond to an input.

```
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
```

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
```

```
sw = swn.senti_synsets('escape')
for item in sw:
    print(item)
```

```
<escape.n.01: PosScore=0.0 NegScore=0.0>
<escape.n.02: PosScore=0.0 NegScore=0.375>
<evasion.n.03: PosScore=0.0 NegScore=0.125>
<escape.n.04: PosScore=0.375 NegScore=0.375>
<escape.n.05: PosScore=0.0 NegScore=0.0>
<escape.n.06: PosScore=0.0 NegScore=0.0>
<escape.n.07: PosScore=0.0 NegScore=0.0>
<safety_valve.n.01: PosScore=0.0 NegScore=0.0>
<escape.v.01: PosScore=0.0 NegScore=0.0>
<miss.v.09: PosScore=0.0 NegScore=0.0>
<get_off.v.05: PosScore=0.25 NegScore=0.25>
<elude.v.02: PosScore=0.0 NegScore=0.125>
<escape.v.05: PosScore=0.0 NegScore=0.0>
<scat.v.01: PosScore=0.0 NegScore=0.0>
<escape.v.07: PosScore=0.0 NegScore=0.0>
```

```
sent2="The refugees braved a harrowing journey across the sea to victoriously reach a new lan
tokens2 = word_tokenize(sent2)
for i in tokens2:
```

```
  syn_list = list(swn.senti_synsets(i))
  if syn_list:
    print(syn_list[0])
```

```
    <refugee.n.01: PosScore=0.0 NegScore=0.0>
    <weather.v.01: PosScore=0.0 NegScore=0.625>
    <angstrom.n.01: PosScore=0.0 NegScore=0.0>
    <harrow.v.01: PosScore=0.0 NegScore=0.0>
    <journey.n.01: PosScore=0.0 NegScore=0.0>
    <across.r.01: PosScore=0.0 NegScore=0.0>
    <sea.n.01: PosScore=0.0 NegScore=0.0>
    <victoriously.r.01: PosScore=0.25 NegScore=0.0>
    <range.n.02: PosScore=0.25 NegScore=0.0>
    <angstrom.n.01: PosScore=0.0 NegScore=0.0>
    <new.a.01: PosScore=0.375 NegScore=0.0>
    <land.n.01: PosScore=0.0 NegScore=0.0>
```

Collocations are how often specific words appear together to create a meaning, in the sense that they can be considered a phrase. This is important because it is not purely logical and words with equivalent definitions cannot be swapped out for words in a collocation.

```
from nltk.book import text4
print(text4.collocations())
text = ' '.join(text4.tokens)
```

```
    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
```

```
import math
vocab = len(set(text4))
hg = text.count('Old World')/vocab
h = text.count('Old')/vocab
g = text.count('World')/vocab
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
    pmi =  8.983886091037398
```

```
The collocation "Old World" had a probability
98. This means that that collocation had a hig
actual collocation, and the two words come tog
meaning
```

The collocation "Old World" had a probability of mutual information of 8.98. This means that that

```
meaning.
```

collocation had a high incidence rate and is an actual collocation, and the two words come together to create a unique meaning.