What is WordNet?

WordNet is a database that contains English words, their definitions, examples, the context in which theyre used, and how words can be linked together. In this notebook we use WordNet and its features to analyze words and their synonyms as well as the relations between them.

In [1]:
```
#import WordNet
from nltk.corpus import wordnet as wn
```

Selecting a noun and output all the synsets

In [2]:
```
wn.synsets('home')
```

Out[2]:
```
[Synset('home.n.01'),
 Synset('dwelling.n.01'),
 Synset('home.n.03'),
 Synset('home_plate.n.01'),
 Synset('base.n.14'),
 Synset('home.n.06'),
 Synset('home.n.07'),
 Synset('family.n.01'),
 Synset('home.n.09'),
 Synset('home.v.01'),
 Synset('home.v.02'),
 Synset('home.a.01'),
 Synset('home.a.02'),
 Synset('home.s.03'),
 Synset('home.r.01'),
 Synset('home.r.02'),
 Synset('home.r.03')]
```

Selecting a synset from list above and extract its definition

In [3]:
```
wn.synset('home.n.01').definition()
```

Out[3]:
```
'where you live at a particular time'
```

Extracting an example of how the synset is used

In [4]:
```
wn.synset('home.n.01').examples()
```

Out[4]:
```
['deliver the package to my home',
 "he doesn't have a home to go to",
 'your place or mine?']
```

Displaying the lemmas for the synset

In [5]:
```
wn.synset('home.n.01').lemmas()
```

Out[5]:
```
[Lemma('home.n.01.home'), Lemma('home.n.01.place')]
```

Traversing up the WordNet hierarchy and outputting the synsets

In [6]:
```
home = wn.synset('home.n.01')
home_hypernyms = lambda s: s.hypernyms()
list(home.closure(home_hypernyms))
```

Out[6]:
```
[Synset('residence.n.01'),
 Synset('address.n.02'),
 Synset('geographic_point.n.01'),
 Synset('point.n.02'),
 Synset('location.n.01'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

In [7]:
```
print(wn.synset('address.n.01').hypernyms())
print(wn.synset('address.n.01').hyponyms())
```

```
[Synset('code.n.03')]
[Synset('argument.n.05'), Synset('url.n.01')]
```

Nouns are organized in a hierarchical manner using hypernyms and hyponyms. Hyponyms are the lower form of a word, and hypernym is the higher form of a word. You can picture a pyramid, there are many words at the bottom and as you climb, words begin to fall under the same synset. Below is an illustration. One observation from above code is that in the list of synsets, the second element of address nouns was chosen. After looking at the first element of address noun, it is clear why it is not chosen for hierarchical path of home synset.

In [8]:
```
#an example to illustrate the hierarchical organization of nouns (hyponym of residence
hyper = home.hypernyms()              #home = wn.synset('home.n.01')
print("Hypernyms of home:\n",hyper)  #hypernyms of home

example = wn.synset('residence.n.01') #residence synset
ex_hypo = example.hyponyms()          #hyponym of residence
print('\n',"Hyponyms of residence:\n", ex_hypo)
```

```
Hypernyms of home:
 [Synset('residence.n.01')]

 Hyponyms of residence:
 [Synset('domicile.n.01'), Synset('home.n.01')]
```

Next, we look at hypernyms, hyponyms, meronyms, holonyms, and antonyms

In [9]:
```
home.hypernyms()
```

Out[9]:
```
[Synset('residence.n.01')]
```

In [10]:
```
home.hyponyms()
```

Out[10]:
```
[Synset('home_away_from_home.n.01')]
```

In [11]:
```
home.part_meronyms()
```

Out[11]:
```
[]
```

In [12]:
```
home.part_holonyms()
```

Out[12]:
```
[]
```

In [13]:
```
home = wn.synsets('home', pos=wn.VERB)[0]
home.lemmas()[0].antonyms()
```

Out[13]:
```
[]
```

In [14]:
```
home = wn.synsets('home', pos=wn.NOUN)[0]
home.lemmas()[0].antonyms()
```

Out[14]:
```
[]
```

In [15]:
```
home = wn.synsets('home', pos=wn.ADJ)[0]
home.lemmas()[0].antonyms()
```

Out[15]:
```
[Lemma('away.a.02.away')]
```

Selecting a verb and output all the synsets

In [16]:
```
wn.synsets("run")
```

Out[16]:
```
[Synset('run.n.01'),
 Synset('test.n.05'),
 Synset('footrace.n.01'),
 Synset('streak.n.01'),
 Synset('run.n.05'),
 Synset('run.n.06'),
 Synset('run.n.07'),
 Synset('run.n.08'),
 Synset('run.n.09'),
 Synset('run.n.10'),
 Synset('rivulet.n.01'),
 Synset('political_campaign.n.01'),
 Synset('run.n.13'),
 Synset('discharge.n.06'),
 Synset('run.n.15'),
 Synset('run.n.16'),
 Synset('run.v.01'),
 Synset('scat.v.01'),
 Synset('run.v.03'),
 Synset('operate.v.01'),
 Synset('run.v.05'),
 Synset('run.v.06'),
```

```
 Synset('function.v.01'),
 Synset('range.v.01'),
 Synset('campaign.v.01'),
 Synset('play.v.18'),
 Synset('run.v.11'),
 Synset('tend.v.01'),
 Synset('run.v.13'),
 Synset('run.v.14'),
 Synset('run.v.15'),
 Synset('run.v.16'),
 Synset('prevail.v.03'),
 Synset('run.v.18'),
 Synset('run.v.19'),
 Synset('carry.v.15'),
 Synset('run.v.21'),
 Synset('guide.v.05'),
 Synset('run.v.23'),
 Synset('run.v.24'),
 Synset('run.v.25'),
 Synset('run.v.26'),
 Synset('run.v.27'),
 Synset('run.v.28'),
 Synset('run.v.29'),
 Synset('run.v.30'),
 Synset('run.v.31'),
 Synset('run.v.32'),
 Synset('run.v.33'),
 Synset('run.v.34'),
 Synset('ply.v.03'),
 Synset('hunt.v.01'),
 Synset('race.v.02'),
 Synset('move.v.13'),
 Synset('melt.v.01'),
 Synset('ladder.v.01'),
 Synset('run.v.41')]
```

Selecting a synset from list above and extract its definition

In [17]:
```
run_syn = wn.synset('run.v.01')
run_syn.definition()
```

Out[17]:
```
"move fast by using one's feet, with one foot off the ground at any given time"
```

Extracting an example of how the synset is used

In [18]:
```
run_syn.examples()
```

Out[18]:
```
["Don't run--you'll be out of breath", 'The children ran to the store']
```

Displaying the lemmas for the synset

In [19]:
```
run_syn.lemmas()
```

Out[19]:
```
[Lemma('run.v.01.run')]
```

Traversing up the WordNet hierarchy and outputting the synsets

In [20]:
```
run_hypernyms = lambda x: x.hypernyms()
print(list(run_syn.closure(run_hypernyms)))
print(run_syn.hypernyms())
print(run_syn.root_hypernyms())    #top of tree
```

```
[Synset('travel_rapidly.v.01'), Synset('travel.v.01')]
[Synset('travel_rapidly.v.01')]
[Synset('travel.v.01')]
```

An observation of the verbs tree is that it is MUCH shorter of a list than that of the noun tree.
Another is that for all the synsets of "run" lead to a different root, as opposed to noun tree.

In [21]:
```
print(wn.synset('run.v.01').root_hypernyms())
print(wn.synset('run.v.13').root_hypernyms())
print(wn.synset('run.v.14').root_hypernyms())
```

```
[Synset('travel.v.01')]
[Synset('function.v.01')]
[Synset('change.v.02')]
```

For noun tree, all the synsets of home lead to the same root.

In [22]:
```
print(wn.synset('home.n.01').root_hypernyms())
print(wn.synset('home.n.03').root_hypernyms())
print(wn.synset('home.n.09').root_hypernyms())
```

```
[Synset('entity.n.01')]
[Synset('entity.n.01')]
[Synset('entity.n.01')]
```

Using Morphy to find as many different forms of the word "run"

In [23]:
```
print(wn.morphy('run', wn.VERB))
print(wn.morphy('run', wn.NOUN))
print(wn.morphy('run', wn.ADJ))
```

```
run
run
None
```

Picking two words that might be similar and choosing a synset for each word

In [24]:
```
print(wn.synsets("baby"), "\n")
print(wn.synsets("lady"))
```

```
[Synset('baby.n.01'), Synset('baby.n.02'), Synset('child.n.03'), Synset('baby.n.04'), Sy
nset('baby.n.05'), Synset('baby.n.06'), Synset('baby.n.07'), Synset('pamper.v.01')]

[Synset('lady.n.01'), Synset('dame.n.02'), Synset('lady.n.03')]
```

Observing the Wu-Palmer similarity metric and the Lesk algorithm

In [25]:
```
baby = wn.synset('baby.n.01')
lady = wn.synset('lady.n.01')
wn.wup_similarity(baby, lady)
```

Out[25]:
```
0.5714285714285714
```

In [26]:
```
from nltk.wsd import lesk
from nltk import word_tokenize
sentence = "My baby sure is a pretty young lady!"
tokens = word_tokenize(sentence)
print(lesk(sentence, 'lady', 'n'))
print(lesk(sentence, 'baby', 'n'))
```

```
Synset('lady.n.03')
Synset('baby.n.07')
```

SentiWordNet is a lexicon contained in WordNet that has positive and negative category scores for each word in the database. It is used to determine the sentiment associated with a word or sentence of words. You could use this information to determine meaning behind text. Was the sender of a message beign sarcastic or facetious?

Selecting an emotionally charged word, finding its senti-synsets, and outputting the polarity scores for each word.

In [27]:
```
from nltk.corpus import sentiwordnet as swn
slist = (swn.senti_synsets('furious'))
for element in slist:
    print(element)
```

```
<ferocious.s.01: PosScore=0.25 NegScore=0.25>
<angered.s.01: PosScore=0.25 NegScore=0.25>
<angry.s.02: PosScore=0.375 NegScore=0.5>
```

Making up a sentence and outputting the polarity for each word in the sentence

In [32]:
```
sentence = "I had a very busy weekend"
neg = 0
pos = 0
tokens = sentence.split()
for token in tokens:
    slist = list(swn.senti_synsets(token))
    if slist:
        synset = slist[0]
        neg += synset.neg_score()
        pos += synset.pos_score()
print("Negative: ", neg, "\n")
print("Positive: ", pos)
```

```
Negative:  0.0

Positive:  1.125
```

After altering words in the sentence, i noticed the scores didnt reflect the way i was meaning for the

sentence to be understood. For example, I replaced busy with a bad and still got a value greater than zero for positive score.

If these scores were a litte more correct this feature would be great for text communication. When you dont get to see the other person, their message may not be recieved as intended, but it would be interesting for something like to be able to show how the text was intended to come across. To sense sarcasm in text or possibly a persons feelings, like where slight physical cues would give it away, as in a high-pitched voice and disagreement.

A **Collocation** is when you put words together to amplify your meaning. An example could be a textbook definition. Together the words mean that the object/action being described is/sounds to proper or elegant that it could've came from a published textbook. Separately, one could possibly confuse the meaning to either the definition of the textbook or a definition from a textbook. The phrase moreso meaning to be a simile or metaphor.

Outputting collocations for text4, the inaugural address

In [29]:
```
from nltk.book import *
text4.collocations()
```

```
*** 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
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
```

Selecting one of the collocations identified by NLTK and calculating mutual information

In [30]:
```
import math
text = ' '.join(text4.tokens)
words = len(set(text4))
xy = text.count('fellow citizens') / words
print("p(fellow citizens) = ", xy)

x = text.count('fellow') / words
print("p(fellow) = ", x)

y = text.count('citizens') / words
print('p(citizens) = ', y)
```

```
pmi = math.log2(xy / (x * y))
print('pmi = ', pmi)
```

```
p(fellow citizens) =  0.006084788029925187
p(fellow) =  0.013665835411471322
p(citizens) =  0.026932668329177057
pmi =  4.0472042737811735
```

Write commentary on the results of the mutual information formula and your interpretation

The probability of either word, fellow or citizen, being a collocation has a value below 1. But the probability of the words combined has a value much greater than 1, with respect to the other values. This of course could be due to that fact that to be a collocation, you need to have two words minimum.

In [ ]: