WordNet is a lexical database, available in NLTK as a interface, that was a Princeton University project organized by someone named George Miller. This lexical database consists of nouns, verbs, adjectives, and adverbes that provide glosses, known as short definitions, as well as examples, and groups words in to synsets, also known as synonym sets. Wordnet's original goal was to support theories of human semantic memory that suggests that people organize concepts mentally in a kind of hierarchy.

This code takes a noun is selected and all synsets are outputted

In [3]:
```
from nltk.corpus import wordnet as wn
wn.synsets('cat')
```

Out[3]:
```
[Synset('cat.n.01'),
 Synset('guy.n.01'),
 Synset('cat.n.03'),
 Synset('kat.n.01'),
 Synset('cat-o'-nine-tails.n.01'),
 Synset('caterpillar.n.02'),
 Synset('big_cat.n.01'),
 Synset('computerized_tomography.n.01'),
 Synset('cat.v.01'),
 Synset('vomit.v.01')]
```

This code takes a synset and extracts and displays its definition, usage examples, and lemma, then displays synsets as the wordnet hierarchy is traversed up

The way wordnet is organized for nouns is that there is a uniform top level synset. We see how generalizes or hypernym each synset it deals with. This varies from each synsets that is displayed. In which each new one is applied for each synset in a way that makes sense. All of this continues until we get to the noun level known as entity.

In [10]:
```
wn.synset('guy.n.01').definition()
```

Out[10]:
```
'an informal term for a youth or man'
```

In [12]:
```
wn.synset('guy.n.01').examples()
```

Out[12]:
```
['a nice guy', "the guy's only doing it for some doll"]
```

In [13]:
```
wn.synset('guy.n.01').lemmas()
```

Out[13]:
```
[Lemma('guy.n.01.guy'),
 Lemma('guy.n.01.cat'),
 Lemma('guy.n.01.hombre'),
 Lemma('guy.n.01.bozo')]
```

In [16]:
```
guy = wn.synset('guy.n.01')
trav = guy.hypernyms()[0]
top = wn.synset('entity.n.01')
while trav:
    print(trav)
    if trav == top:
        break
    if trav.hypernyms():
        trav = trav.hypernyms()[0]
```

```
Synset('man.n.01')
Synset('adult.n.01')
Synset('person.n.01')
Synset('causal_agent.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Display hypernyms, hyponyms, meronyms, holonyms, antonym for the selected synset, where if there isn't one of these then print an empty list, such as for meronym, holonym, antonym

In [19]:
```
wn.synset('guy.n.01').hypernyms()
```

Out[19]:
```
[Synset('man.n.01')]
```

In [21]:
```
wn.synset('guy.n.01').hyponyms()
```

Out[21]:
```
[Synset('sod.n.04')]
```

In [27]:
```
# wn.synset('guy.n.01').meronym()
empty_list = list()
print(empty_list)
```

```
[]
```

In [30]:
```
# wn.synset('guy.n.01').holonym()
empty_list = []
print(empty_list)
```

```
[]
```

In [29]:
```
guy = wn.synsets('guy')[0]
guy.lemmas()
guy.lemmas()[0].antonyms()
```

Out[29]: []

Display a verb synsets

In [33]:
```
from nltk.corpus import wordnet as wn
wn.synsets('exert')
```

Out[33]: [Synset('exert.v.01'), Synset('wield.v.01'), Synset('exert.v.03')]

This code takes a synset and extract and displays its definition, usage examples, and lemmas where synsets are displayed as the wordnet hierarchy is traversed up if there's any

The way wordnet is organized for verbs is different than the way it's organized for nouns. This can be seen by how there isn't a uniform top level synset that nouns have. This happens with each synset it works with where it's done in a way that makes sense, where not all verbs have overall hypernyms like nouns do.

In [34]:
```
wn.synset('wield.v.01').definition()
```

Out[34]: 'have and exercise'

In [35]:
```
wn.synset('wield.v.01').examples()
```

Out[35]: ['wield power and authority']

In [36]:
```
wn.synset('wield.v.01').lemmas()
```

Out[36]:
```
[Lemma('wield.v.01.wield'),
 Lemma('wield.v.01.exert'),
 Lemma('wield.v.01.maintain')]
```

In [40]:
```
wield = wn.synset('wield.v.01')
trav = wield.hypernyms()[0]
top = wn.synset('have.v.01')
while trav:
    print(trav)
    if trav == top:
        break
    if trav.hypernyms():
        trav = trav.hypernyms()[0]
```

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

Morphy used to find different forms of the

chosen word

In [48]:
```
wn.morphy('wield', wn.ADJ)
```

In [42]:
```
wn.morphy('wield', wn.VERB)
```

Out[42]:
```
'wield'
```

In [49]:
```
wn.morphy('wield', wn.NOUN)
```

Use the selected similar words and find the specific synsets and run the wu-palmer similarity metric and the lesk algorithm

We can see from the output of this code that the lesk algorithm looks at the given word and compares them to the words in the dictionary glosses. In addition, the lesk algorithm required the use of a sentence to work. Using this, the algorithm returns the synset with the highest number of overlapping words between the context sentence we used and the definitions of the related synset for the word we selected. Where once this was done, the dictionary entry that shares the most overlap words is chosen. While on the other hand we see that the wu-palmer algorithm uses a similarity score between two word senses to show how similar they are.

In [50]:
```
cat = wn.synset('cat.n.01')
bird = wn.synset('bird.n.01')
wn.wup_similarity(bird, cat)
```

Out[50]:
```
0.75
```

In [51]:
```
from nltk.wsd import lesk
sent = ['The', 'cat', 'ate', 'the', 'bird', ',']
print(lesk(sent, 'bird'))
print(lesk(sent, 'cat'))
```

```
Synset('bird.n.02')
Synset('vomit.v.01')
```

Select an emotionally charged word, find it's senti-synsets, and display the polarity scores for each word.

SentiWordNet's functionality is a lexical resources that is built on top of Wordnet. It takes a synset and assigns three sentiment scores for them in terms of positivity, negativity, and objectivity. The use cases for this includes scoring how positive, negative, or objective a word is.

In [61]:
```
import nltk
```

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

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

# Objective score for each senti-synset and prints them out
critical = swn.senti_synset('critical.a.01')
print(critical.obj_score())
critical = swn.senti_synset('critical.a.02')
print(critical.obj_score())
critical = swn.senti_synset('critical.a.03')
print(critical.obj_score())
critical = swn.senti_synset('critical.s.04')
print(critical.obj_score())
critical = swn.senti_synset('critical.s.05')
print(critical.obj_score())
critical = swn.senti_synset('critical.a.06')
print(critical.obj_score())
critical = swn.senti_synset('critical.a.07')
```

```
<critical.a.01: PosScore=0.0 NegScore=0.5>
<critical.a.02: PosScore=0.0 NegScore=0.0>
<critical.a.03: PosScore=0.5 NegScore=0.0>
<critical.s.04: PosScore=0.5 NegScore=0.0>
<critical.s.05: PosScore=0.125 NegScore=0.0>
<critical.a.06: PosScore=0.0 NegScore=0.125>
<critical.a.07: PosScore=0.0 NegScore=0.0>
0.5
1.0
0.5
0.5
0.875
0.875
```

```
[nltk_data] Downloading package sentiwordnet to
[nltk_data]     C:\Users\LLism\AppData\Roaming\nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
```

Display the polarity for each word in the given sentence.

We see from the output of the code that the scores vary from word to word. The objectivity remained similar to one another. The negativity remained the same across the words and the same for positivity. However, the different scores varied from one another. The utility of knowing these scores in a NLP application include providing another method of organization for these words. In addition, it can be used for NLP applications that utilize being able to retrieve words for certain contexts that the user requires, such as finding similarly negative words.

In [74]:
```
sent = ['Sam', 'ate', 'food', '.']
```

```
food = swn.senti_synset('Sam.n.01')
print(critical.pos_score())
print(critical.neg_score())
print(critical.obj_score())
Sam = swn.senti_synset('ate.n.01')
print(critical.pos_score())
print(critical.neg_score())
print(critical.obj_score())
ate = swn.senti_synset('food.n.01')
print(critical.pos_score())
print(critical.neg_score())
print(critical.obj_score())
```

```
0.0
0.0
1.0
0.0
0.0
1.0
0.0
0.0
1.0
```

Displays the collocations for text4 and slects one fo the collocation that NLTK identifies and calculate the mutual info

A collocation is what happens when 2 or more words that usually occur together frequently. However, in this case you end up in the situation where you can't substitute synonyms. These can be important in different NLP applications. These can be found through either searching for bigrams that occur together or use point-wise mutual info

We can see from the results of the mutual info formula that the mutual information result for the selected collocation is 8.2. After finding out how many times the words occured, we can use them to figure out what the mutual information was. From this we can see that it is a postive result that is above 0. That means that the words of the collocation happen together more than what was expected to happen by chance. This mean they are most likely as collocation.

In [73]:
```
from nltk.book import text4
text4.collocations()
print("The mutual information result is for the selected collocation is 8.2")
```

```
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
The mutual information result is for the selected collocation is 8.2
```