(1)

```
import nltk
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
```

(2) Get the synsets of 'hand'

```
synsets = wn.synsets('hand')
print([l.name() for l in synsets])
```

```
['hand.n.01', 'hired_hand.n.01', 'handwriting.n.01', 'hand.n.04', 'hand.n.05', 'hand.n.
```

(3) Get the definition, usage, and lemmas of the 'hand.n.01' synset. Get the hierarchy of synsets for 'hand.n.01'.

The word 'hand' has quite a few synsets. I selected 'hand.n.01' because I wanted the physical body part. Hand is categorized 8 layers down in the hierarchical structure of nouns as a hyponym of 'extremity.n.05'. The biggest thing to note about the noun organizational hierarchy is that the top is always 'entity.n.01'. An entity is the highest overarching form for all nouns.

```
print('Definition: ', synsets[0].definition())
print('Examples:   ', synsets[0].examples())
print('Lemmas:     ', [l.name() for l in synsets[0].lemmas()])
print('Hierarchy:  ', [l.name() for l in synsets[0].closure(lambda s:s.hypernyms()
```

```
Definition:  the (prehensile) extremity of the superior limb
Examples:    ['he had the hands of a surgeon', 'he extended his mitt']
Lemmas:      ['hand', 'manus', 'mitt', 'paw']
Hierarchy:   ['extremity.n.05', 'external_body_part.n.01', 'body_part.n.01', 'part.n.03
```

(4) Get the Hypernyms, Hyponyms, Meronyms, Holonyms, and Antonyms of 'hand.n.01'.

```
print('Hypernyms:  ', [l.name() for l in synsets[0].hypernyms()])
print('Hyponyms:   ', [l.name() for l in synsets[0].hyponyms()])
print('Meronyms:   ', [l.name() for l in synsets[0].part_meronyms()])
print('Holonyms:   ', [l.name() for l in synsets[0].part_holonyms()])
print('Antonyms:   ', [l.name() for l in synsets[0].lemmas()[0].antonyms()])
```

```
Hypernyms:   ['extremity.n.05']
Hyponyms:    ['fist.n.01', 'hooks.n.01', 'left.n.03', 'right.n.05']
Meronyms:    ['ball.n.10', 'digital_arteries.n.01', 'finger.n.01', 'intercapitular_vein
Holonyms:    ['arm.n.01', 'homo.n.02']
Antonyms:    []
```

(5) Get the synsets of 'cry'

```
synsets2 = wn.synsets('cry', pos=wn.VERB)
print(synsets2)
```

```
[Synset('shout.v.02'), Synset('cry.v.02'), Synset('exclaim.v.01'), Synset('cry.v.04'), S
```

(6) Get the definition, usage, and lemmas of the 'cry.v.02' synset. Get the hierarchy of synsets for 'cry.v.02'.

The word 'cry' has fewer synsets compared to 'hand'. I selected 'cry.v.02' because it matched the definition I wanted better (tears). 'cry.v.02' actually has no hierarchy above it; it is a top level synset. The biggest thing to note about the verb organizational hierarchy is that there is no specific top level synset. Unlike nouns which end with 'entity.n.01', the ending for a verb hierarchy is unique.

```
print('Definition: ', synsets2[1].definition())
print('Examples:   ', synsets2[1].examples())
print('Lemmas:     ', [l.name() for l in synsets2[1].lemmas()])
print('Hierarchy:  ', [l.name() for l in synsets2[1].closure(lambda s:s.hypernyms())])
```

```
Definition:  shed tears because of sadness, rage, or pain
Examples:    ['She cried bitterly when she heard the news of his death', 'The girl in th
Lemmas:      ['cry', 'weep']
Hierarchy:   ['express_emotion.v.01']
```

(7) Use morphy to test various different forms of 'cry'

```
print(wn.morphy('cry', wn.VERB))
print(wn.morphy('cried', wn.VERB))
print(wn.morphy('cries', wn.VERB))
print(wn.morphy('cries', wn.NOUN))
print(wn.morphy('crying', wn.VERB))
```

```
cry
cry
cry
cry
cry
```

(8) Compare the similarity of the words 'banana' and 'apple'.

The path similarity uses the hierarchical path of the synsets to calculate how similar the words are. It tends to not be very accurate.

The Wu-Palmer similarity looks at common ancestor words to calculate how similar the given words are.

The Lesk algorithm is given the definition of 'banana.n.01' and attempts to find the closest synset of 'apple' given the context. In this case, it actually said that 'apple.n.02' was closest to 'banana.n.01'. After looking at the definitions of both, the similarities are evident. The biggest difference between 'apple.n.01' and 'apple.n.02' is that 'apple.n.02' describes the tree and environment of apples instead of the fruit itself. 'banana.n.01' is closer to 'apple.n.02' because 'banana.n.01' also describes the tree/environment.

```
word1 = 'banana'
word2 = 'apple'

word1Syn = wn.synsets(word1)[0]
print(word1Syn, word1Syn.definition())
word2Syn = wn.synsets(word2)[0]
print(word2Syn, word2Syn.definition())

print('\nPath Similarity:       ', wn.path_similarity(word1Syn, word2Syn))
print('Wu-Parlmer Similarity: ', word1Syn.wup_similarity(word2Syn))

# Use the definition for word1 to give context to see which synset of word2 is most similar
leskResult = lesk(word1Syn.definition(), word2)
print('\nLesk Algorythm Result: ', leskResult)
print(leskResult, leskResult.definition())
```

```
Synset('banana.n.01') any of several tropical and subtropical treelike herbs of the genu
Synset('apple.n.01') fruit with red or yellow or green skin and sweet to tart crisp whit

Path Similarity:        0.07142857142857142
Wu-Parlmer Similarity:  0.38095238095238093

Lesk Algorythm Result:  Synset('apple.n.02')
Synset('apple.n.02') native Eurasian tree widely cultivated in many varieties for its fi
```

(9) SentiWordNet is a lexical resource that assigns sentiment scores for a given synset. The scores are split into 3 categories: (Positivity, Negativity, Objectivity) and their values may range from 0 to 1. (Pos + Neg = 1). To use it, you must input a synset to the function senti_synset() and it will return an object with the positive, negative, and objectivity scores.

Below are various examples of emotionally charged words and their sentiment scores. 'Beautiful' is mostly positive, 'Ugly' is somewhat negative, 'Lustful' is apparently equally positive and negative (neutral?), and resentful is very negative.

This technology can be vary useful for trying to decipher the opinion of a piece of text. Here are some links to papers written about its utility on opinion mining:

https://github.com/aesuli/SentiWordNet/blob/master/papers/LREC06.pdf

https://github.com/aesuli/SentiWordNet/blob/master/papers/LREC10.pdf

```
def getSynsets(word):
  print('Synsets for', word + ':')
  syns = wn.synsets(word)
  print('  ', [(l.name(), l.definition()) for l in syns])
  return syns

def calcSentiment(syn):
  print('Selected: ', syn.name())
  senti = swn.senti_synset(syn.name())
  print(f'  Positive: {senti.pos_score()}, Negative: {senti.neg_score()}, Objective: {senti.o

syns = getSynsets('beautiful')
calcSentiment(syns[0])

syns = getSynsets('ugly')
calcSentiment(syns[0])

syns = getSynsets('lustful')
calcSentiment(syns[1])

syns = getSynsets('resentful')
calcSentiment(syns[0])
```

```
Synsets for beautiful:
   [('beautiful.a.01', 'delighting the senses or exciting intellectual or emotional admi
Selected:  beautiful.a.01
  Positive: 0.75, Negative: 0.0, Objective: 0.25

Synsets for ugly:
   [('ugly.a.01', 'displeasing to the senses'), ('surly.s.01', 'inclined to anger or bac
Selected:  ugly.a.01
  Positive: 0.0, Negative: 0.375, Objective: 0.625

Synsets for lustful:
   [('lubricious.s.02', 'characterized by lust'), ('lascivious.s.01', 'driven by lust; p
Selected:  lascivious.s.01
  Positive: 0.5, Negative: 0.5, Objective: 0.0
```

```
Synsets for resentful:
   [('resentful.a.01', 'full of or marked by resentment or indignant ill will')]
Selected:  resentful.a.01
  Positive: 0.0, Negative: 0.875, Objective: 0.125
```

(10) A collocation is when a set of words frequently occur together within a piece of text. A way to determine whether a set of words is a collocation or not is to attempt to substitute a word for one of its synonyms. If the overall meaning changes then it is a collocation, if not, then it may simply be coincidental or a language quirk. In NLP it is important to understand collocations because they may imbue certain meanings to the text outside of their individual definitions.

A way to determine if a collocation is real or coincidental is through the point wise mutual information formula. The formula is simply the probability of the collocation divided by the product of the probability of the individual words. A score of 0 means the words are independent of each other. A positive score means they are likely a collocation while a negative score means it is unlikely.

The example below uses the Inaugural Address Corpus. The words 'fellow citizens' got a score of 8.06 which means it is likely to be a collocation.

```
import math

print(text4.name)
text4Colloc = text4.collocations()

text = ' '.join([t.lower() for t in text4.tokens])
vocabSize = len(set(text4))

fc = text.count('fellow citizens') / vocabSize
f  = text.count('fellow') / vocabSize
c  = text.count('citizens') / vocabSize

print('\nSelected "fellow citizens"')
print('Mutual Info Score: ', math.log2(fc / (f * c)))
```

```
Inaugural Address Corpus
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

Selected "fellow citizens"
Mutual Info Score:  8.062981491349493
```