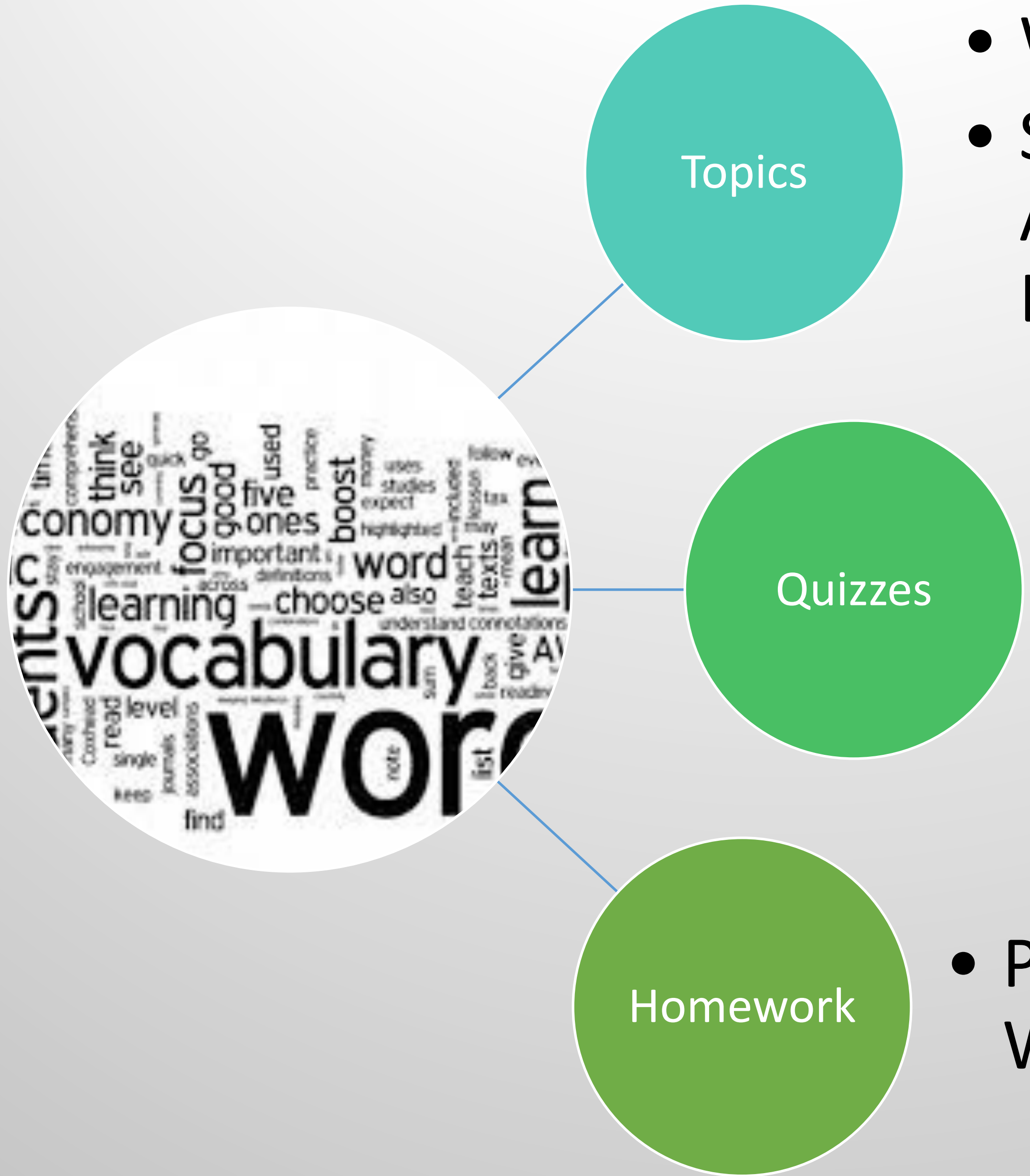
Natural Language Processing

Dr. Karen Mazidi

Part Two: Words



Topics

- WordNet
- Sentiment Analysis with Lexicons

Quizzes

- No quiz on this material

Homework

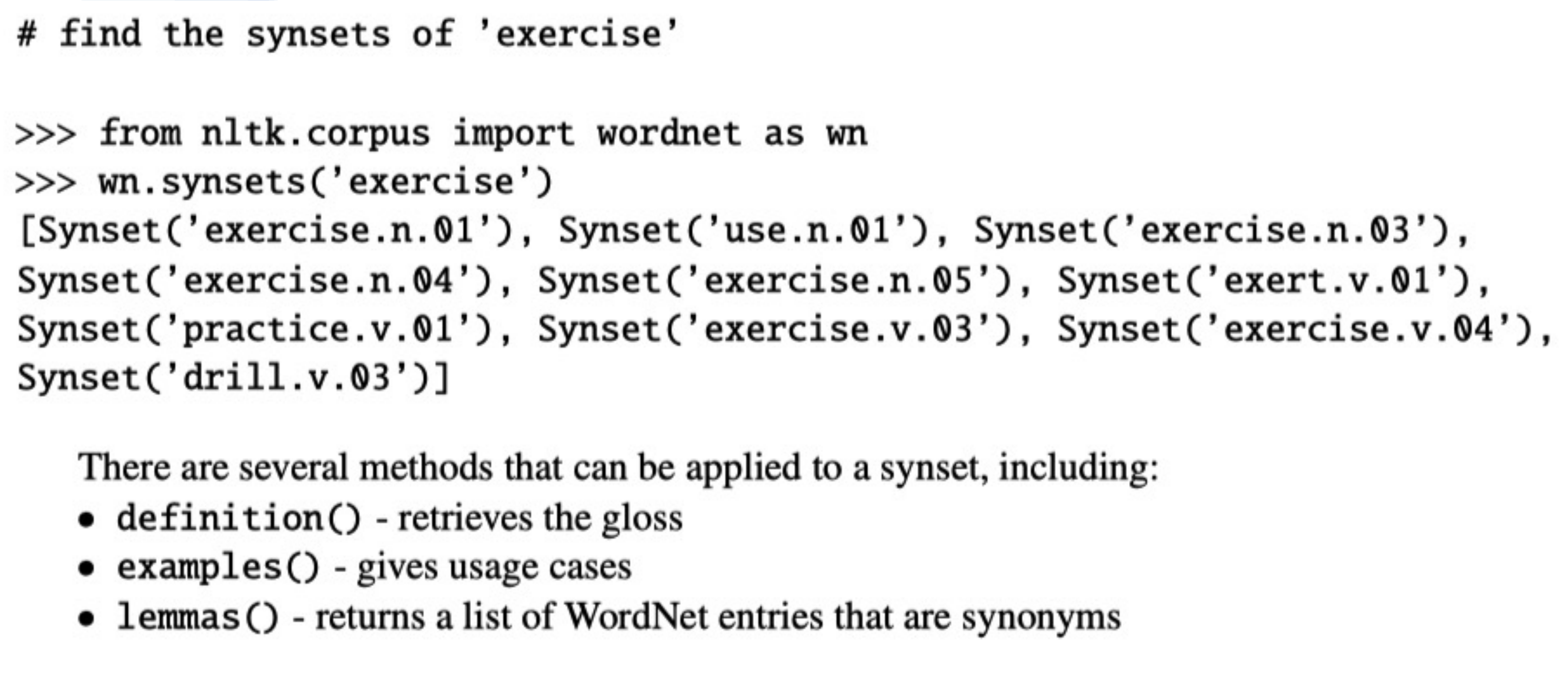
- Portfolio: WordNet

WordNet

- WordNet is a project started at Princeton in the mid 1980s by the psychologist George Miller, who was interested in how people hierarchically organize concepts
- https://wordnet.princeton.edu/
- WordNet is a hierarchical organization of nouns, verbs, adjectives and adverbs; listing:
  - Glosses: short definitions
  - Synsets: synonym sets
  - Use examples
  - Relations to other words

WordNet
in
NLTK

Explore a synset
(Notebook 7.1)



```
# find the synsets of 'exercise'

>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('exercise')
[Synset('exercise.n.01'), Synset('use.n.01'), Synset('exercise.n.03'),
Synset('exercise.n.04'), Synset('exercise.n.05'), Synset('exert.v.01'),
Synset('practice.v.01'), Synset('exercise.v.03'), Synset('exercise.v.04'),
Synset('drill.v.03')]
```

There are several methods that can be applied to a synset, including:

- `definition()` - retrieves the gloss
- `examples()` - gives usage cases
- `lemmas()` - returns a list of WordNet entries that are synonyms

synset relations

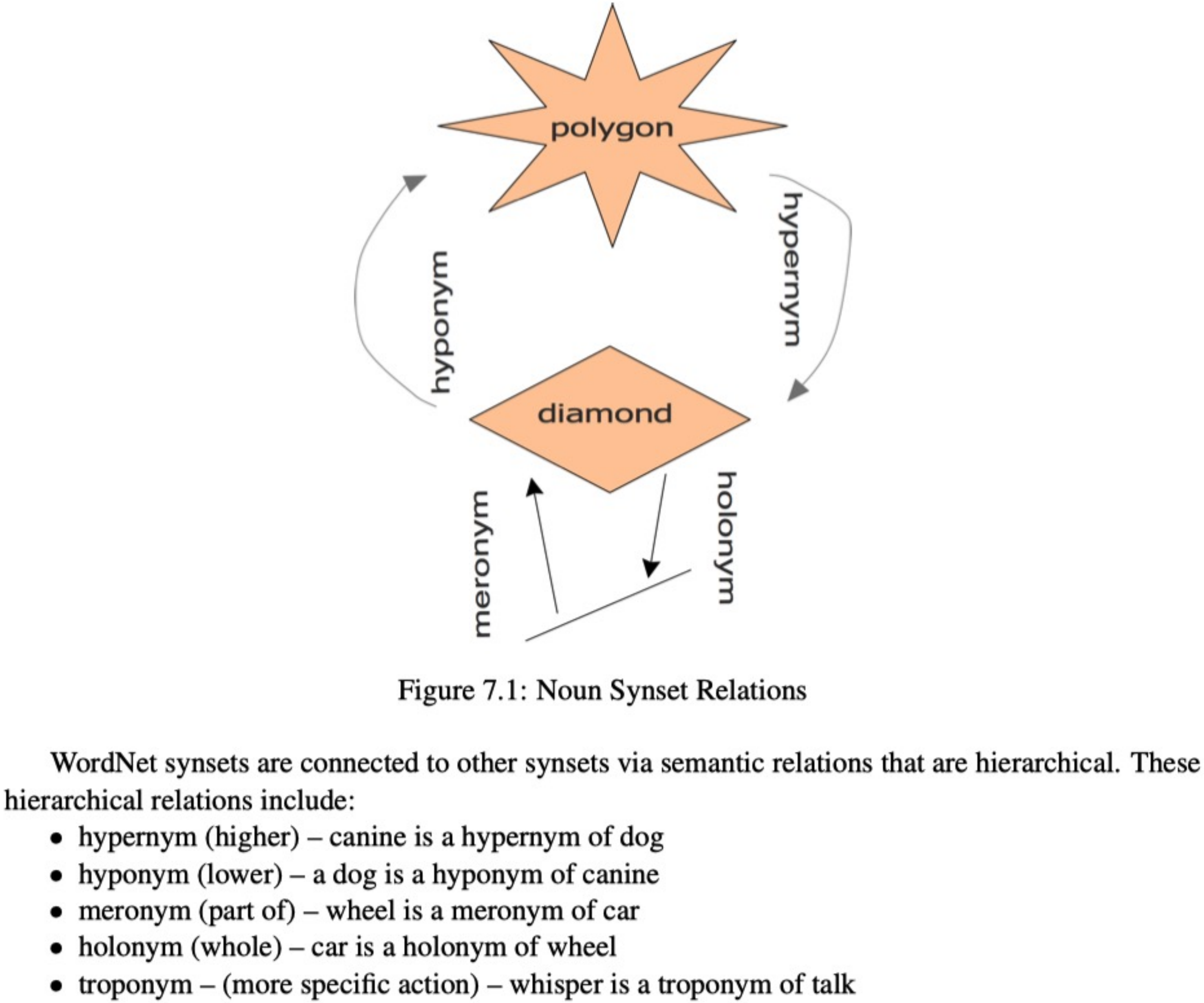

polygon
hyponym
hypernym
diamond
meronym
holonym

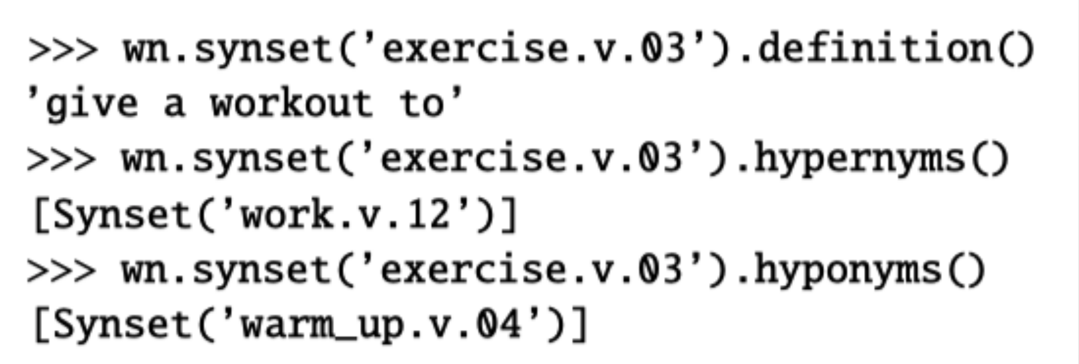
Figure 7.1: Noun Synset Relations

WordNet synsets are connected to other synsets via semantic relations that are hierarchical. These hierarchical relations include:

- hypernym (higher) – canine is a hypernym of dog
- hyponym (lower) – a dog is a hyponym of canine
- meronym (part of) – wheel is a meronym of car
- holonym (whole) – car is a holonym of wheel
- troponym – (more specific action) – whisper is a troponym of talk

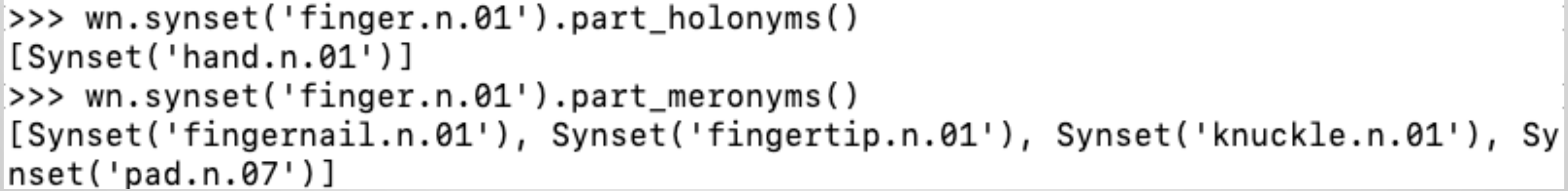
synset relations

- Not every lemma has an entry for every relation
- Nouns are the most highly-connected synsets
- Verbs can be in hypernym/hyponym relations



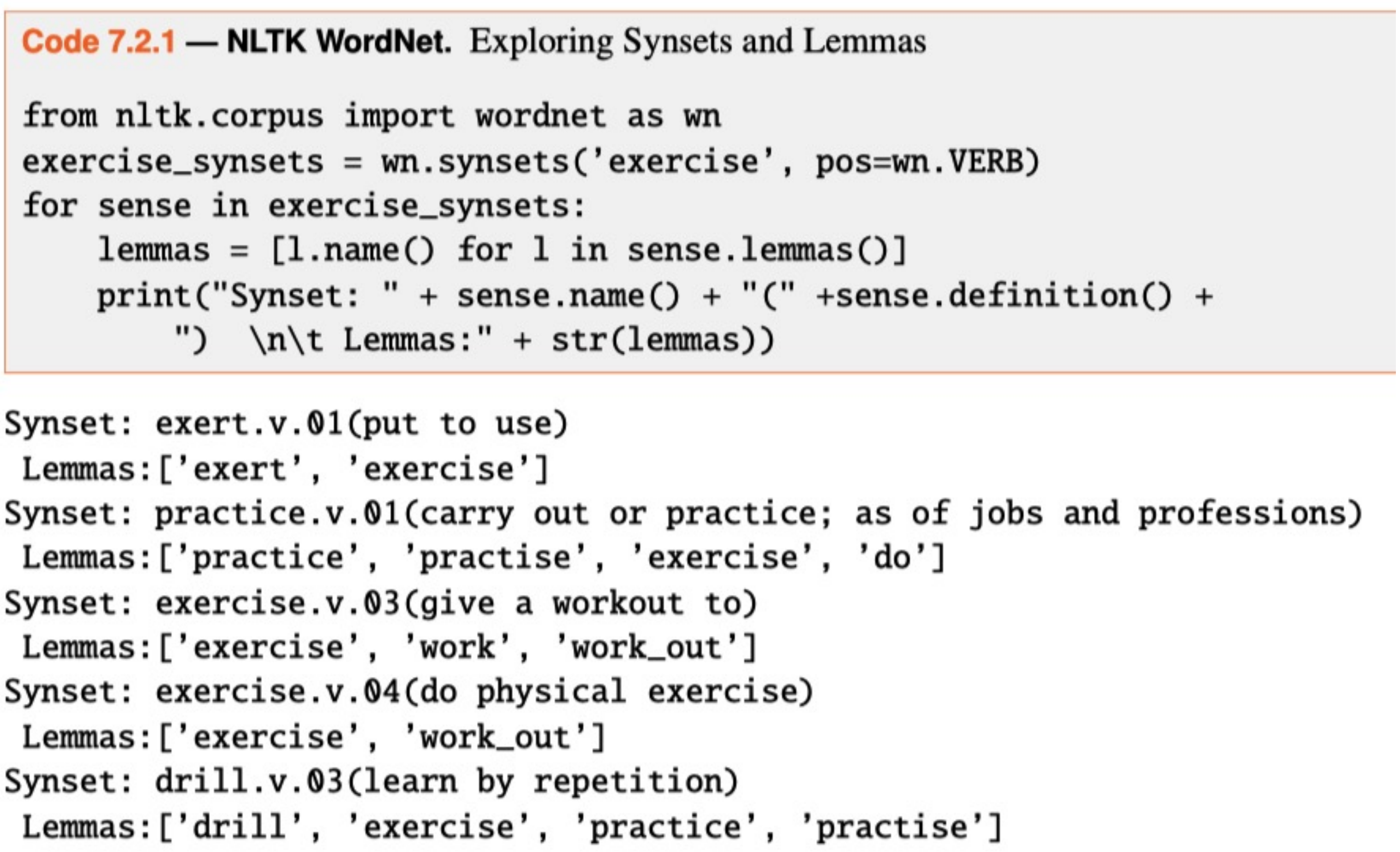
```
>>> wn.synset(’exercise.v.03’).definition()
’give a workout to’
>>> wn.synset(’exercise.v.03’).hypernyms()
[Synset(’work.v.12’)]
>>> wn.synset(’exercise.v.03’).hyponyms()
[Synset(’warm_up.v.04’)]
```

part-whole relations



```
>>> wn.synset('finger.n.01').part_holonyms()
[Synset('hand.n.01')]
>>> wn.synset('finger.n.01').part_meronyms()
[Synset('fingernail.n.01'), Synset('fingertip.n.01'), Synset('knuckle.n.01'), Sy
nset('pad.n.07')]
```

Find lemmas in a synset



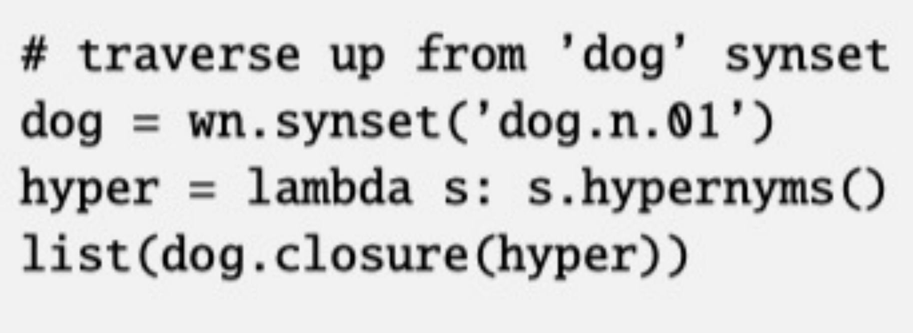
Code 7.2.1 — NLTK WordNet. Exploring Synsets and Lemmas

```
from nltk.corpus import wordnet as wn
exercise_synsets = wn.synsets(’exercise’, pos=wn.VERB)
for sense in exercise_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() +
        ")  \n\t Lemmas:" + str(lemmas))
```

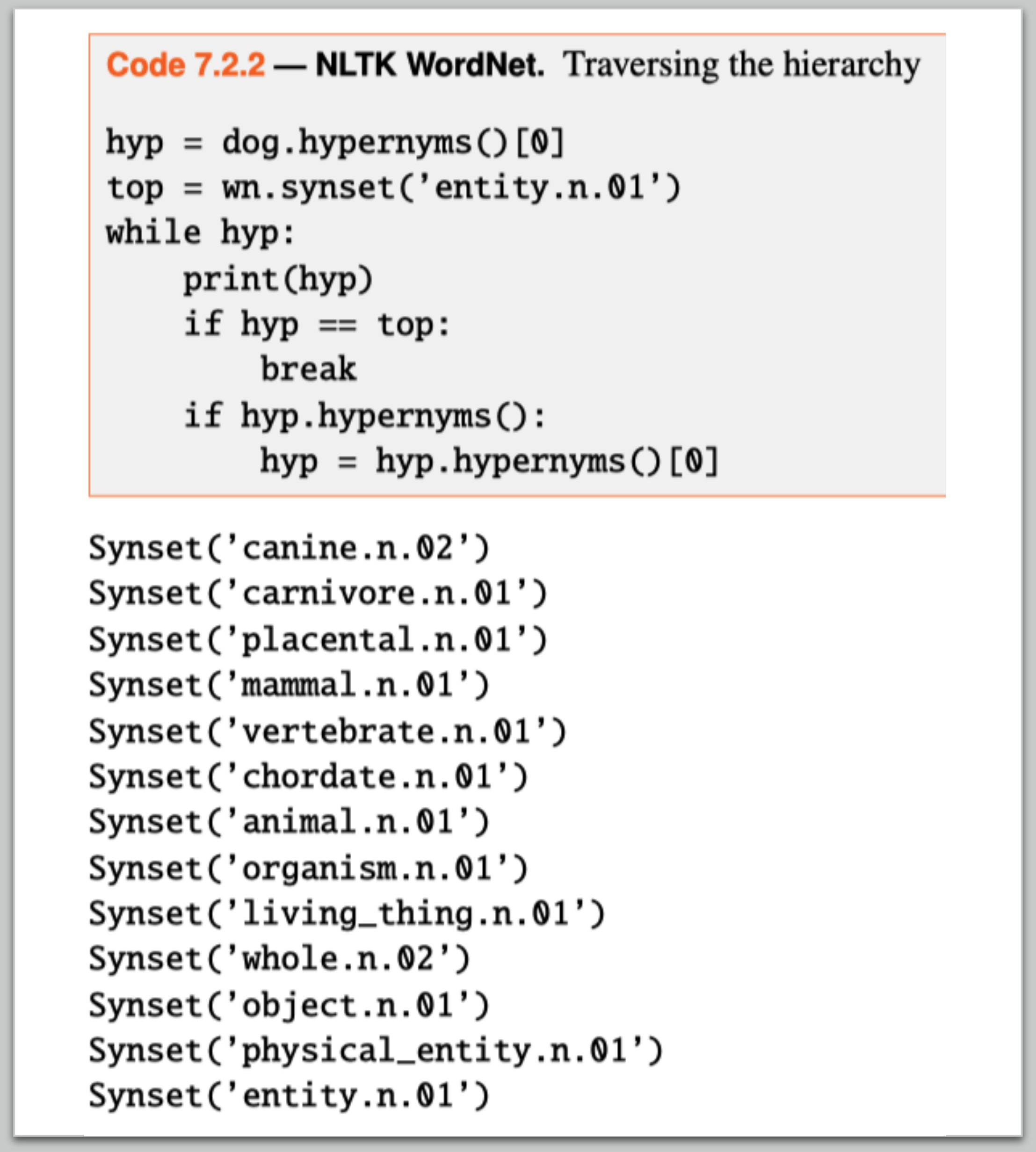
```
Synset: exert.v.01(put to use)
 Lemmas:[’exert’, ’exercise’]
Synset: practice.v.01(carry out or practice; as of jobs and professions)
 Lemmas:[’practice’, ’practise’, ’exercise’, ’do’]
Synset: exercise.v.03(give a workout to)
 Lemmas:[’exercise’, ’work’, ’work_out’]
Synset: exercise.v.04(do physical exercise)
 Lemmas:[’exercise’, ’work_out’]
Synset: drill.v.03(learn by repetition)
 Lemmas:[’drill’, ’exercise’, ’practice’, ’practise’]
```

Traversing the hierarchy

- See Notebook 7.2
- Unlike nouns, there is no top level synset for all verbs



```
# traverse up from 'dog' synset
dog = wn.synset('dog.n.01')
hyper = lambda s: s.hypernyms()
list(dog.closure(hyper))
```



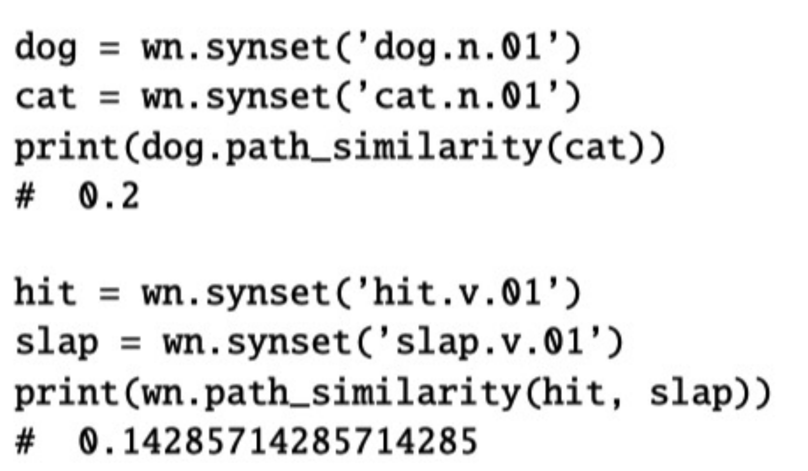
Code 7.2.2 — NLTK WordNet. Traversing the hierarchy

```
hyp = dog.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('canine.n.02')
Synset('carnivore.n.01')
Synset('placental.n.01')
Synset('mammal.n.01')
Synset('vertebrate.n.01')
Synset('chordate.n.01')
Synset('animal.n.01')
Synset('organism.n.01')
Synset('living_thing.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

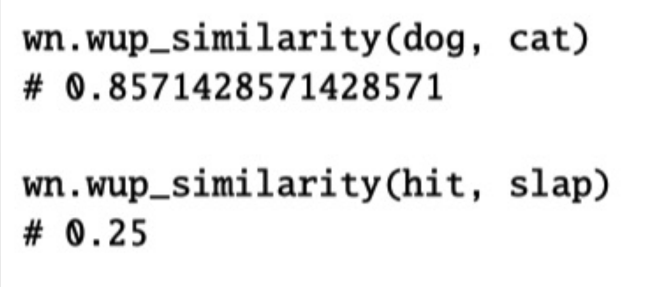
Word similarity

- See notebook 7.3
- Similarity ranges from 0 (little similarity) to 1 (identity)



```
dog = wn.synset(’dog.n.01’)
cat = wn.synset(’cat.n.01’)
print(dog.path_similarity(cat))
#  0.2

hit = wn.synset(’hit.v.01’)
slap = wn.synset(’slap.v.01’)
print(wn.path_similarity(hit, slap))
#  0.14285714285714285
```

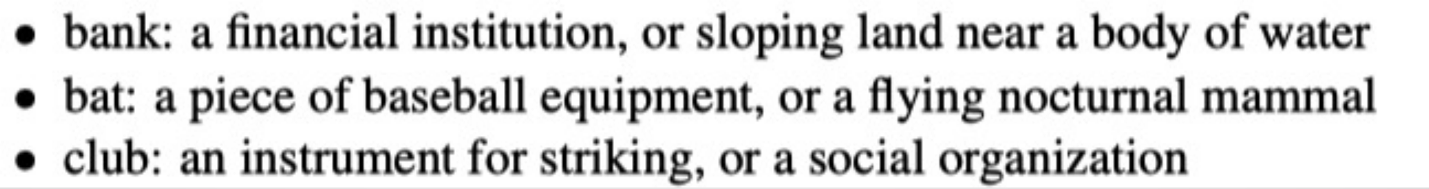
- Compare to Wu-Palmer
- Looks at common path



```
wn.wup_similarity(dog, cat)
# 0.8571428571428571

wn.wup_similarity(hit, slap)
# 0.25
```

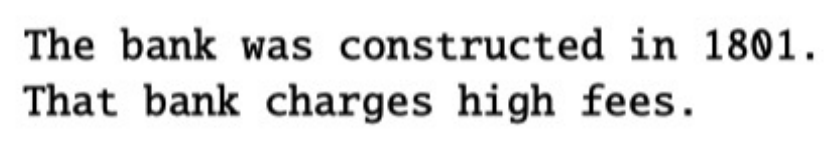
WSD Word Sense Disambiguation

- Homonyms share same word form but have different meanings
- Homophones sound alike but may be spelled differently and have different meanings



- bank: a financial institution, or sloping land near a body of water
- bat: a piece of baseball equipment, or a flying nocturnal mammal
- club: an instrument for striking, or a social organization

- A **polysemous** word has many related meanings, ex: bank can refer to the building or the financial institution



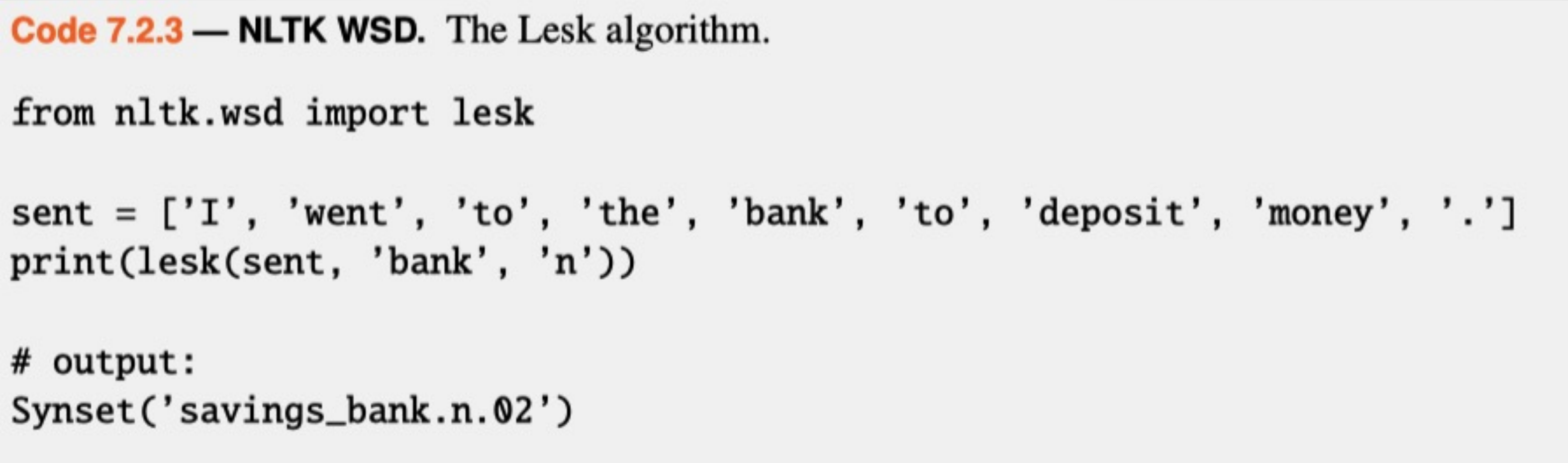
The bank was constructed in 1801.
That bank charges high fees.

WSD

- Metonymy – when a word is substituted for another entity to which it is related
  - ‘suit’ for an executive
  - ‘White House’ for the administration
  - ‘Dallas’ for Dallas Cowboys
- Synecdoche – using part to stand in for the whole thing
  - All hands on deck
  - Nice wheels
- How is an NLP system able to ‘understand’ this figurative speech?

Lesk algorithm

- Looks at context and compares to dictionary glosses for word overlap count



Code 7.2.3 — NLTK WSD. The Lesk algorithm.

```
from nltk.wsd import lesk

sent = [’I’, ’went’, ’to’, ’the’, ’bank’, ’to’, ’deposit’, ’money’, ’.’]
print(lesk(sent, ’bank’, ’n’))

# output:
Synset(’savings_bank.n.02’)
```

what is an adjective satellite?

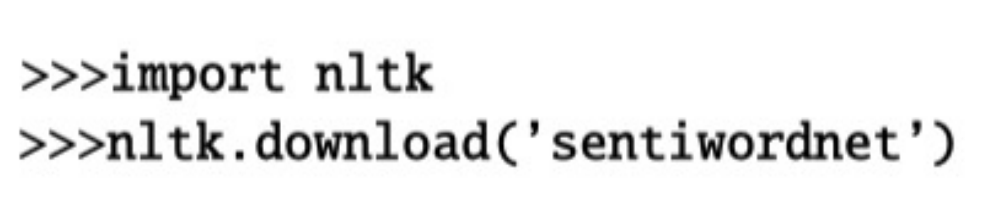
- https://www.englishforums.com/English/AdjectiveSatellite/nwzhv/post.htm
- search for “Synset”

Sentiment Analysis

Using NLP Tools

SentiWordNet

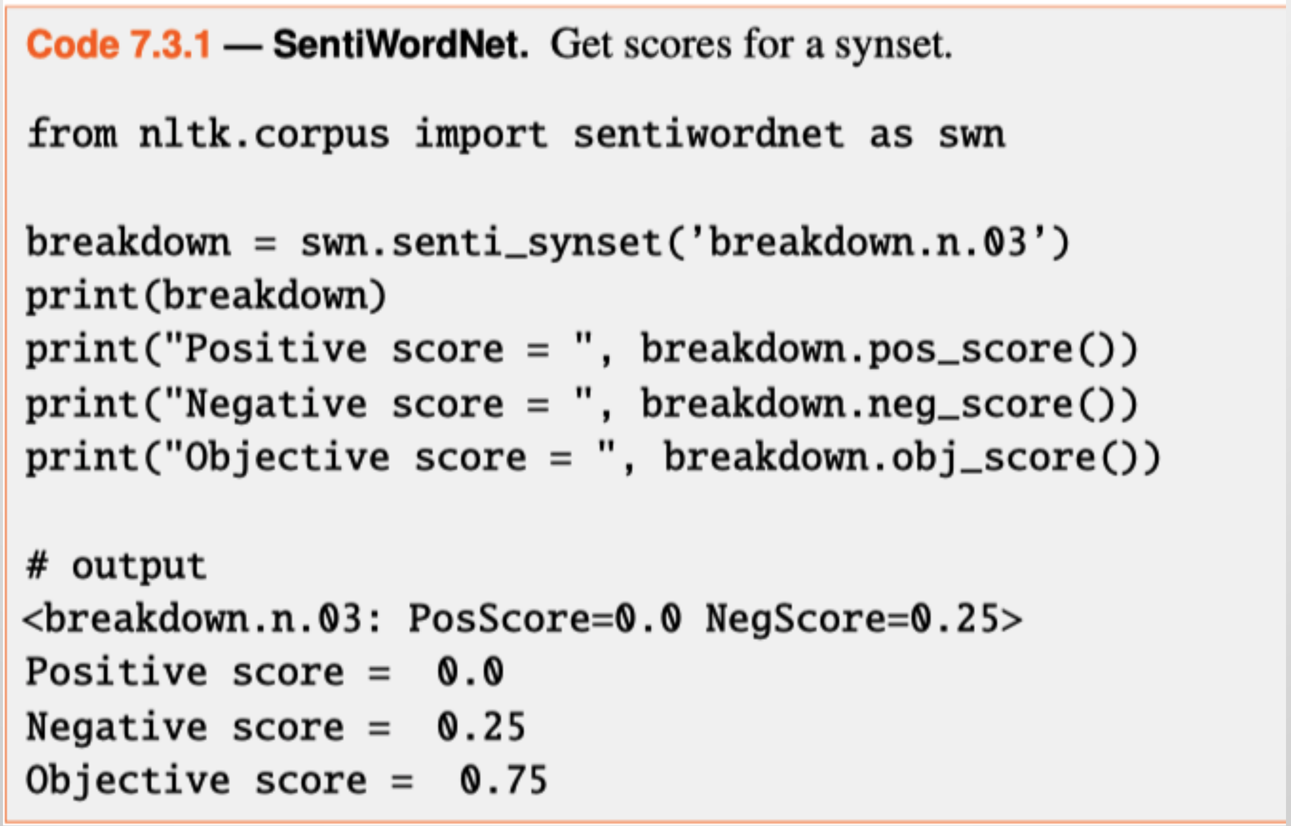
- Built on top of WordNet
- Assigns 3 scores: positive, negative, objectivity
- Corpus needs to be downloaded first



```
>>>import nltk
>>>nltk.download('sentiwordnet')
```

SentiWordNet

- See Notebook 7.3

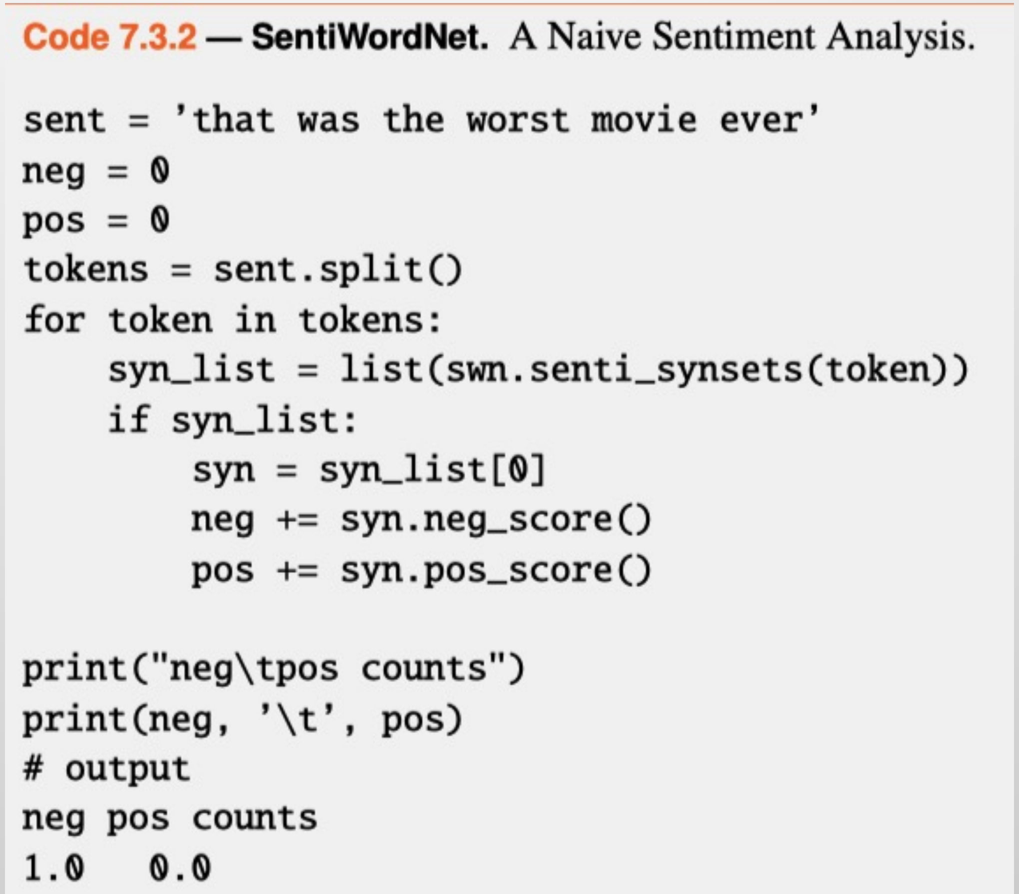


Code 7.3.1 — SentiWordNet. Get scores for a synset.

```
from nltk.corpus import sentiwordnet as swn

breakdown = swn.senti_synset(’breakdown.n.03’)
print(breakdown)
print("Positive score = ", breakdown.pos_score())
print("Negative score = ", breakdown.neg_score())
print("Objective score = ", breakdown.obj_score())

# output
<breakdown.n.03: PosScore=0.0 NegScore=0.25>
Positive score =  0.0
Negative score =  0.25
Objective score =  0.75
```

Sentiment analysis

- Analysis of a text to see if it is positive/negative or objective/subjective



Code 7.3.2 — SentiWordNet. A Naive Sentiment Analysis.

```
sent = ’that was the worst movie ever’
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()

print("neg\tpos counts")
print(neg, ’\t’, pos)
# output
neg pos counts
1.0   0.0
```

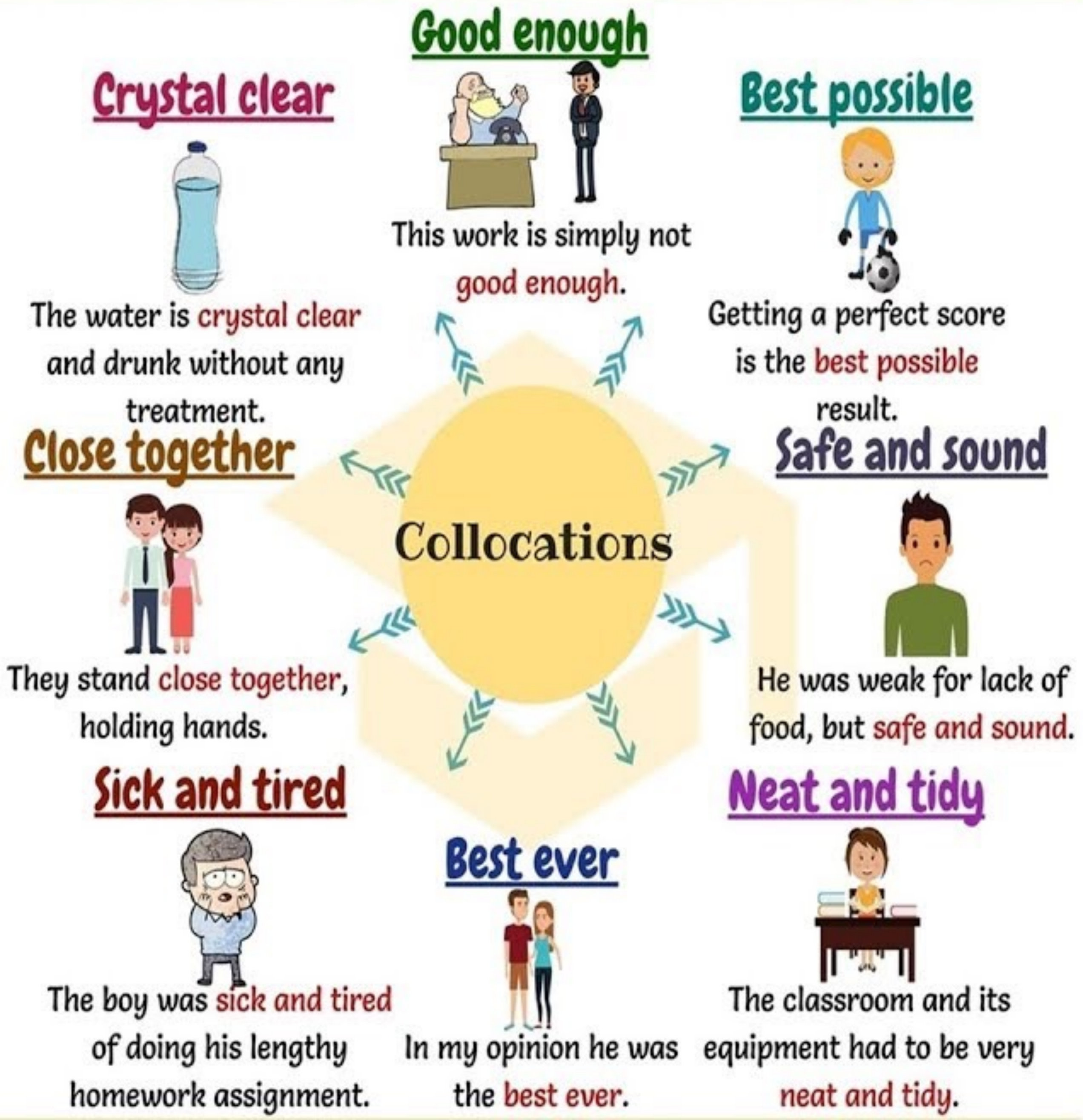
VADER

- Valence Aware Dictionary and sEntiment Reasoner
- pip/pip3 install vaderSentiment
- See Notebook 7.4
- https://github.com/cjhutto/vaderSentiment

collocations

Using NLP Tools

Common Collocations



Collocations

Crystal clear

The water is crystal clear and drunk without any treatment.

Good enough

This work is simply not good enough.

Best possible

Getting a perfect score is the best possible result.

Close together

They stand close together, holding hands.

Safe and sound

He was weak for lack of food, but safe and sound.

Sick and tired

The boy was sick and tired of doing his lengthy homework assignment.

Best ever

In my opinion he was the best ever.

Neat and tidy

The classroom and its equipment had to be very neat and tidy.

ADVERBS + ADVERBS

ADVERBS + ADVERBS

Right now

I'm sorry, but I can't talk right now.

All alone

I was scared because I was all alone.

All along

We knew all along that he was packing a dictionary.

Quite enough

I've had quite enough of your tantrums.

Almost certainly

Almost certainly he will be suited up for the game.

Pretty well

She knows pretty well everything there is to know on the subject.

Only just

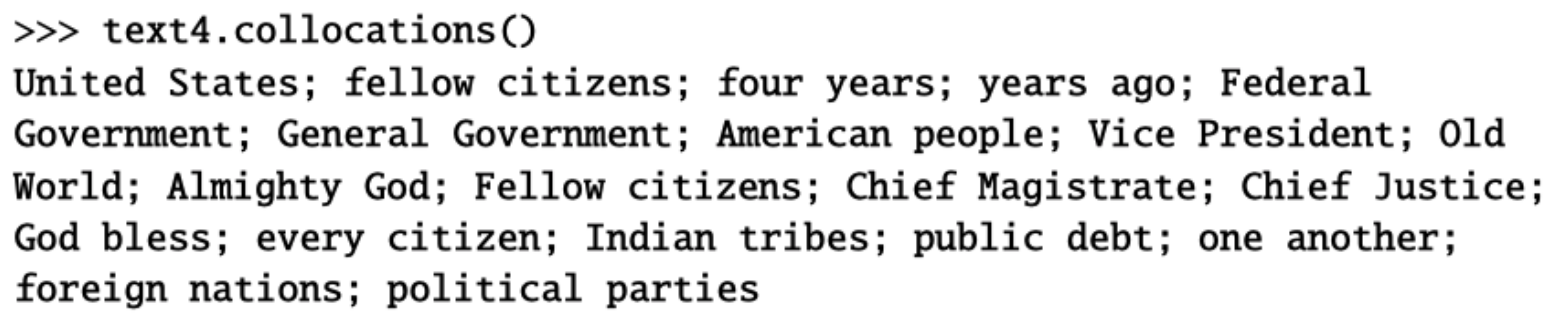
We made it to the airport on time, but only just.

Dead ahead

The school is dead ahead about two miles from here.

collocations

- When two or more words combine more often that expected by chance, and you cannot substitute a word and get the same meaning
  - Wild rice is not unruly rice
  - Strong tea is not muscular tea



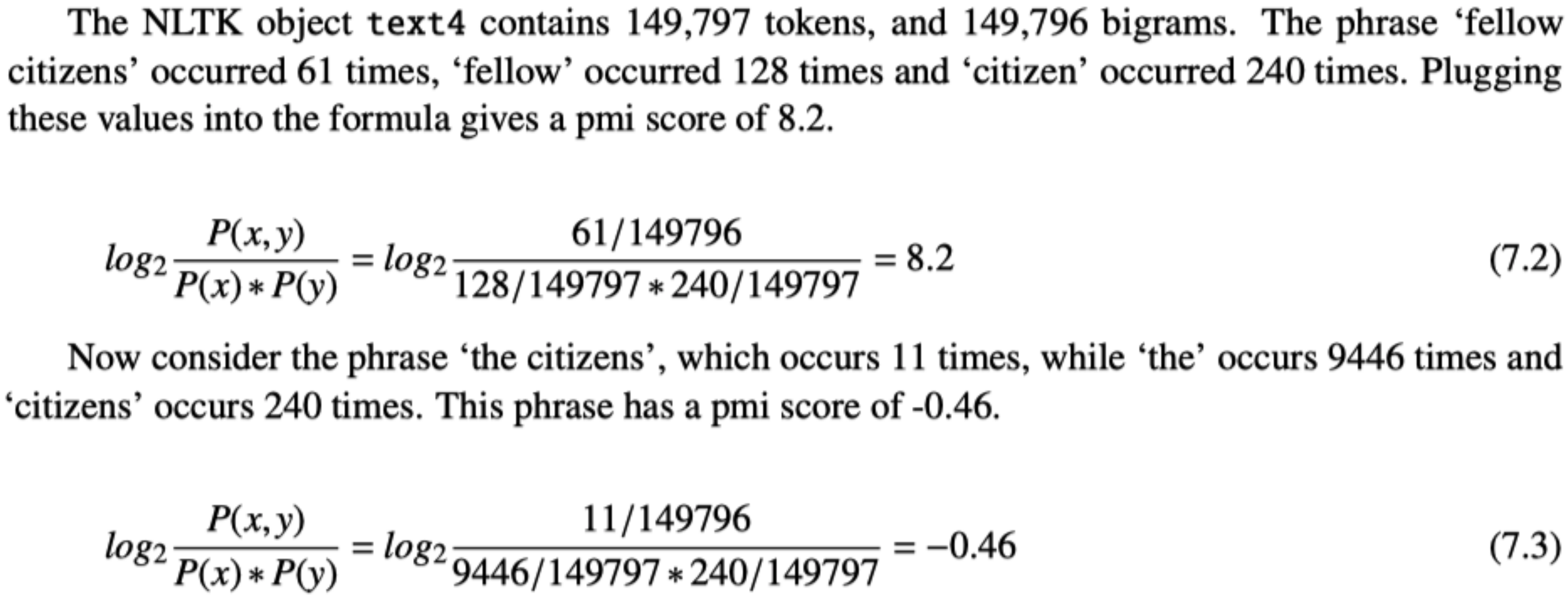
```
>>> text4.collocations()
United States; fellow citizens; four years; years ago; Federal
Government; General Government; American people; Vice President; Old
World; Almighty God; Fellow citizens; Chief Magistrate; Chief Justice;
God bless; every citizen; Indian tribes; public debt; one another;
foreign nations; political parties
```

collocations

- Can be found with pmi (point-wise mutual information)
- PMI of 0 means x and y are independent
- PMI that is positive, then likely to be a collocation
- PMI that is negative, not likely to be a collocation

$$log_2 \frac{P(x,y)}{P(x) * P(y)}$$

Is ‘fellow citizens’ a collocation?



The NLTK object `text4` contains 149,797 tokens, and 149,796 bigrams. The phrase ‘fellow citizens’ occurred 61 times, ‘fellow’ occurred 128 times and ‘citizen’ occurred 240 times. Plugging these values into the formula gives a pmi score of 8.2.

$$log_2 \frac{P(x,y)}{P(x)*P(y)} = log_2 \frac{61/149796}{128/149797*240/149797} = 8.2 \tag{7.2}$$

Now consider the phrase ‘the citizens’, which occurs 11 times, while ‘the’ occurs 9446 times and ‘citizens’ occurs 240 times. This phrase has a pmi score of -0.46.

$$log_2 \frac{P(x,y)}{P(x)*P(y)} = log_2 \frac{11/149796}{9446/149797*240/149797} = -0.46 \tag{7.3}$$

NLP Research

- Related NLP conference:
  SEMEVAL https://semeval.github.io/SemEval2021/tasks.html
- ACL anthology search here: https://aclanthology.org/

WordNet Code Examples

- 7.1 Exploring WordNet synsets
- 7.2 The WordNet hierarchy
- 7.3 SentiWordNet
- 7.4 VADER
- 7.5 Collocations

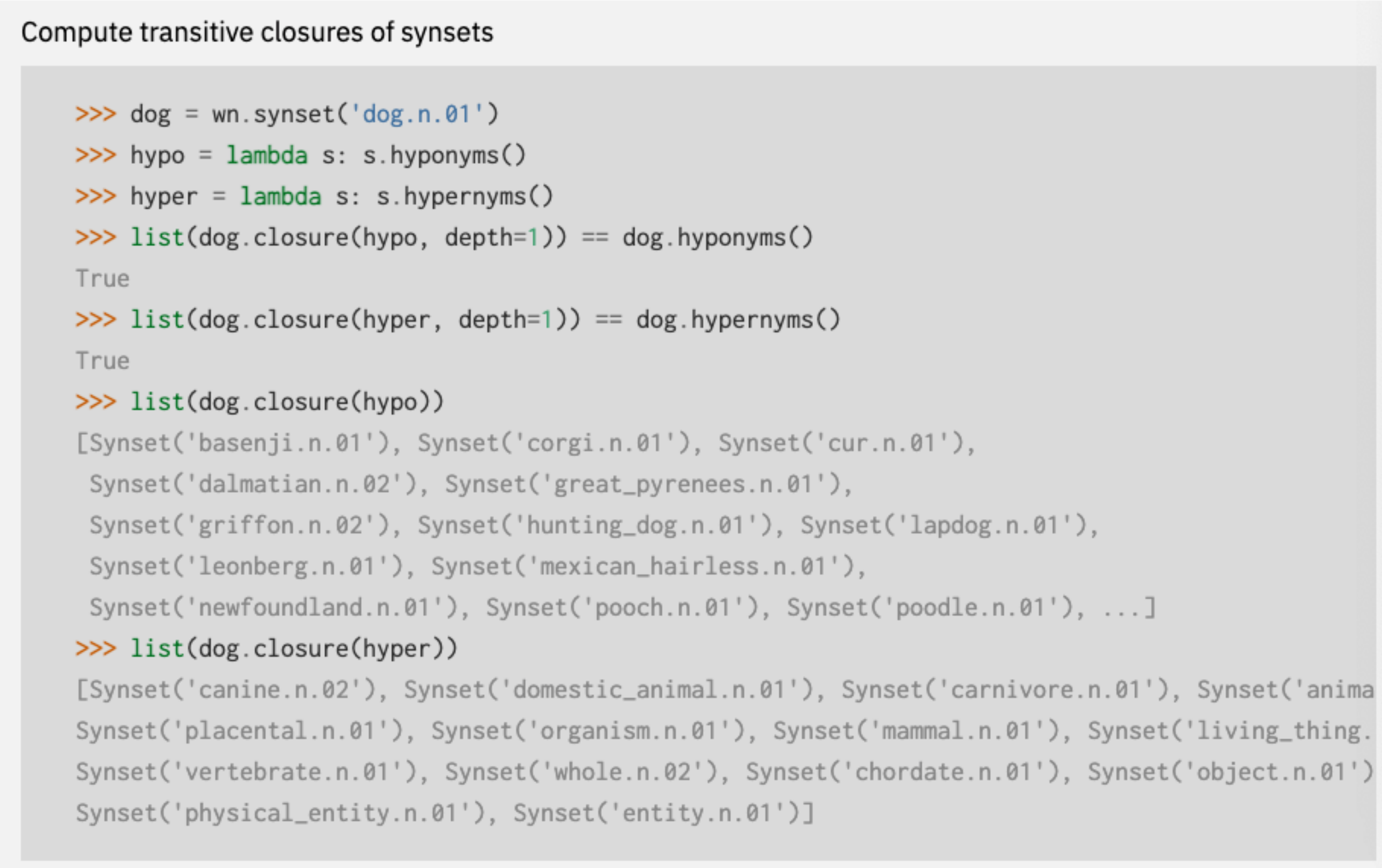
Closures

Extra Python material

Material from:

https://towardsdatascience.com/closures-and-decorators-in-python-2551abbc6eb6

Synset closure in NLTK

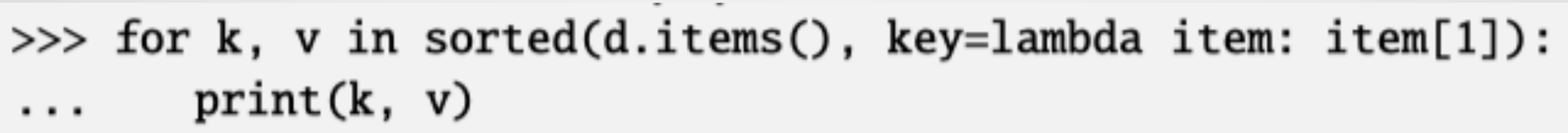


Compute transitive closures of synsets

```
>>> dog = wn.synset('dog.n.01')
>>> hypo = lambda s: s.hyponyms()
>>> hyper = lambda s: s.hypernyms()
>>> list(dog.closure(hypo, depth=1)) == dog.hyponyms()
True
>>> list(dog.closure(hyper, depth=1)) == dog.hypernyms()
True
>>> list(dog.closure(hypo))
[Synset('basenji.n.01'), Synset('corgi.n.01'), Synset('cur.n.01'),
 Synset('dalmatian.n.02'), Synset('great_pyrenees.n.01'),
 Synset('griffon.n.02'), Synset('hunting_dog.n.01'), Synset('lapdog.n.01'),
 Synset('leonberg.n.01'), Synset('mexican_hairless.n.01'),
 Synset('newfoundland.n.01'), Synset('pooch.n.01'), Synset('poodle.n.01'), ...]
>>> list(dog.closure(hyper))
[Synset('canine.n.02'), Synset('domestic_animal.n.01'), Synset('carnivore.n.01'), Synset('anima
Synset('placental.n.01'), Synset('organism.n.01'), Synset('mammal.n.01'), Synset('living_thing.
Synset('vertebrate.n.01'), Synset('whole.n.02'), Synset('chordate.n.01'), Synset('object.n.01')
Synset('physical_entity.n.01'), Synset('entity.n.01')]
```

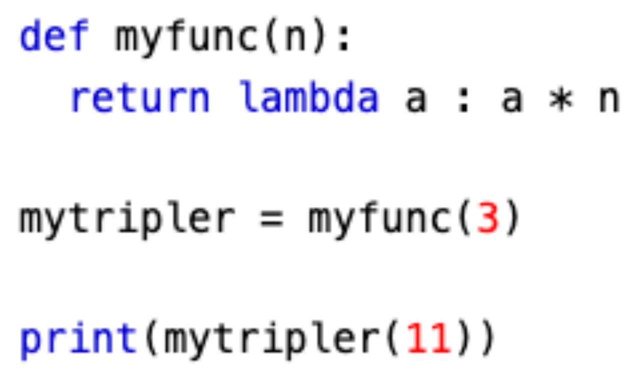
Lambda

- a lambda function is a small anonymous function
- we’ve used it to sort a dict by value



```
>>> for k, v in sorted(d.items(), key=lambda item: item[1]):
...     print(k, v)
```

- below, the function returns a function



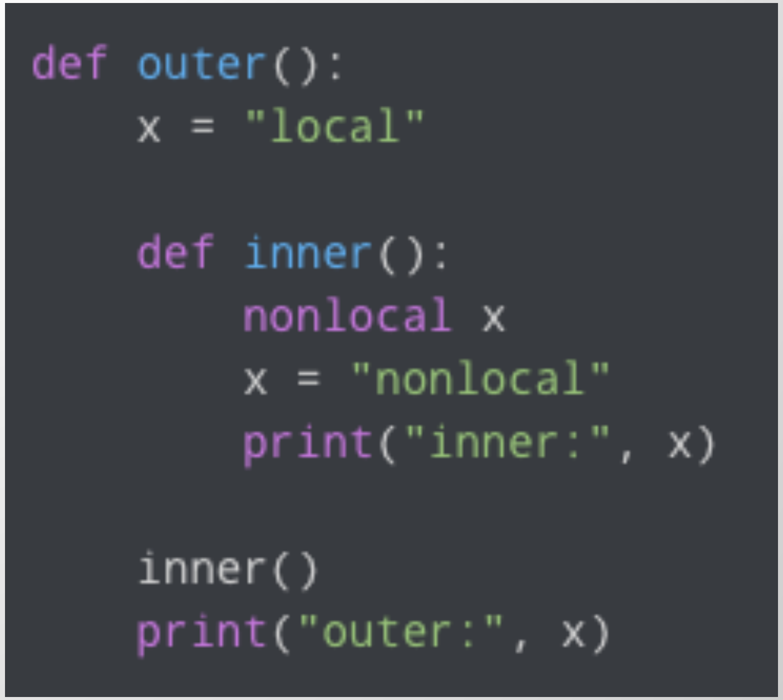
```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

Scope of Python variables

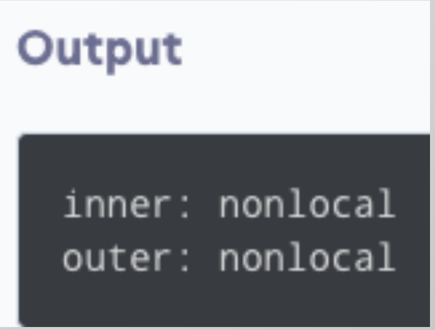
- global scope – variable defined outside all functions
  - can be access by all functions in the file
- local scope – variable defined inside a function
  - can only be accessed inside the function in which it is defined
- nonlocal scope – variable can be accessed by the function in which it was defined and all its nested functions

nonlocal

- a nonlocal variable is neither in local scope nor global scope
- the inner() function changed the value of x, which changed the value of x in the local scope of outer()



```python
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)
```

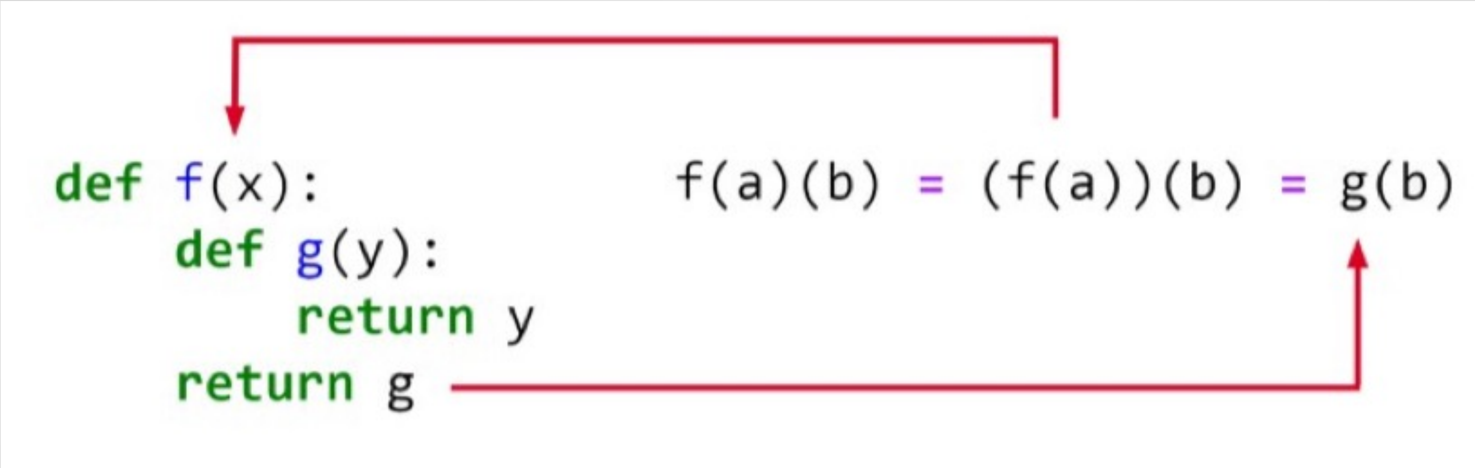


Output

```
inner: nonlocal
outer: nonlocal
```

Python functions

- objects are passed by reference
  - if they are mutable, they can be changed in the function
- Python functions are “first-class” objects, meaning:
  - you can assign a function to a variable
  - you can pass a function as a function argument
  - you can return a function from a function



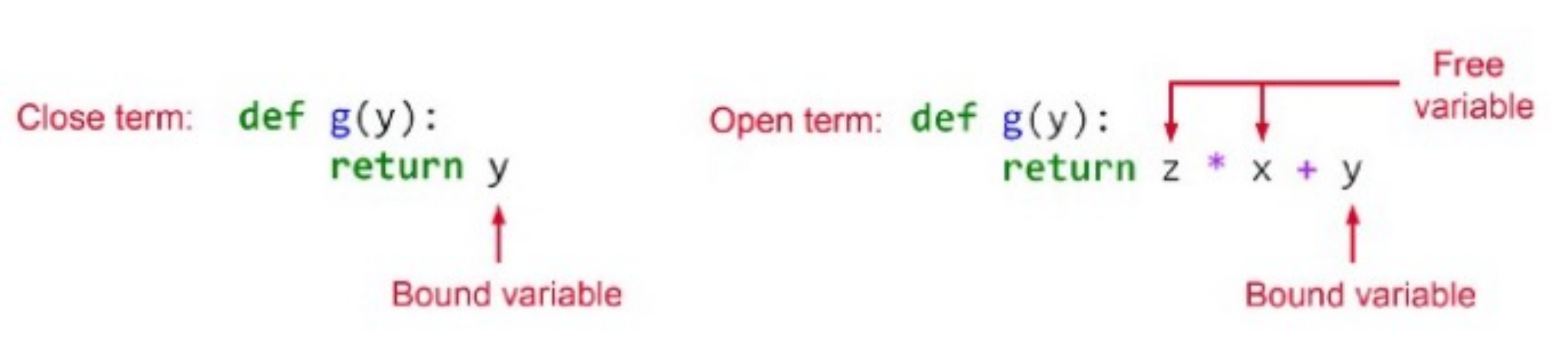
```
def f(x):
    def g(y):
        return y
    return g
```

```
f(a)(b) = (f(a))(b) = g(b)
```

Closure

- a closure is an inner function with an extended scope that encompasses nonlocal variables of the outer function
- a closure remembers the nonlocal variables in the enclosing scopes even if they are not present in memory
- the name ‘closure’ comes from the fact that it captures the bindings of its free (nonlocal) variables and is the result of closing an open term

Closure

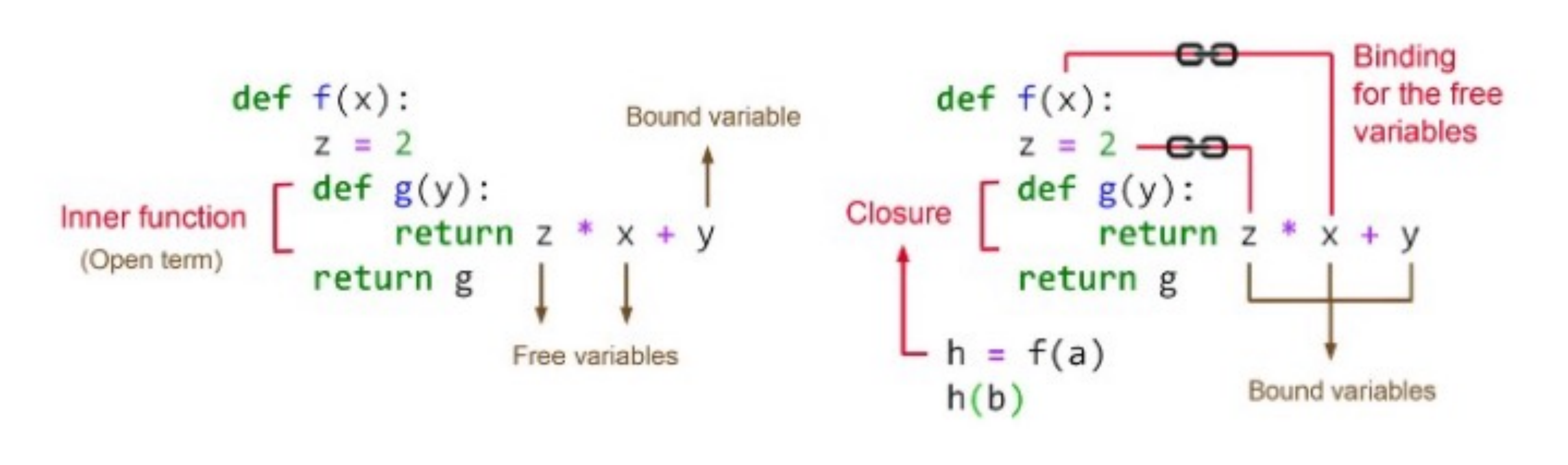


```
def g(y):
    return y
```

```
def g(y):
    return z * x + y
```

- variable y is a bound variable because it is in the local scope of g()
- variables x and z is a free variable because it was defined outside of g()
- a function that contains only bound variables is called a close term
- a function that contains free variables is called an open term
- a closure is an open term which is closed by capturing the bindings of its free (nonlocal) variables

Closure



```
def f(x):
    z = 2
    def g(y):
        return z * x + y
    return g
```

```
def f(x):
    z = 2
    def g(y):
        return z * x + y
    return g

h = f(a)
h(b)
```

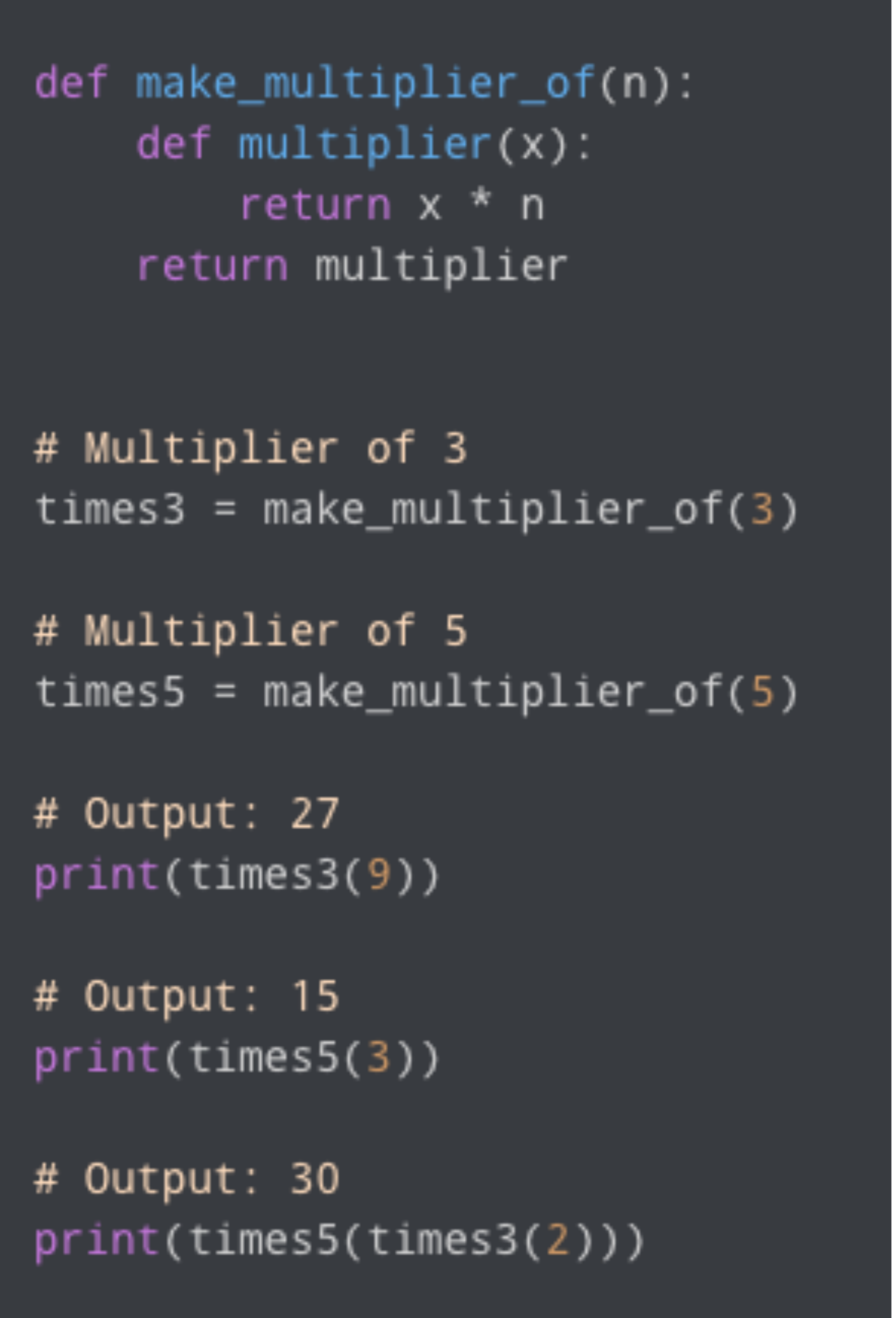
- the inner function must be returned by the outer function
- the inner function should capture some of the nonlocal variables of the outer function
- call the outer function to return the closure

Closures

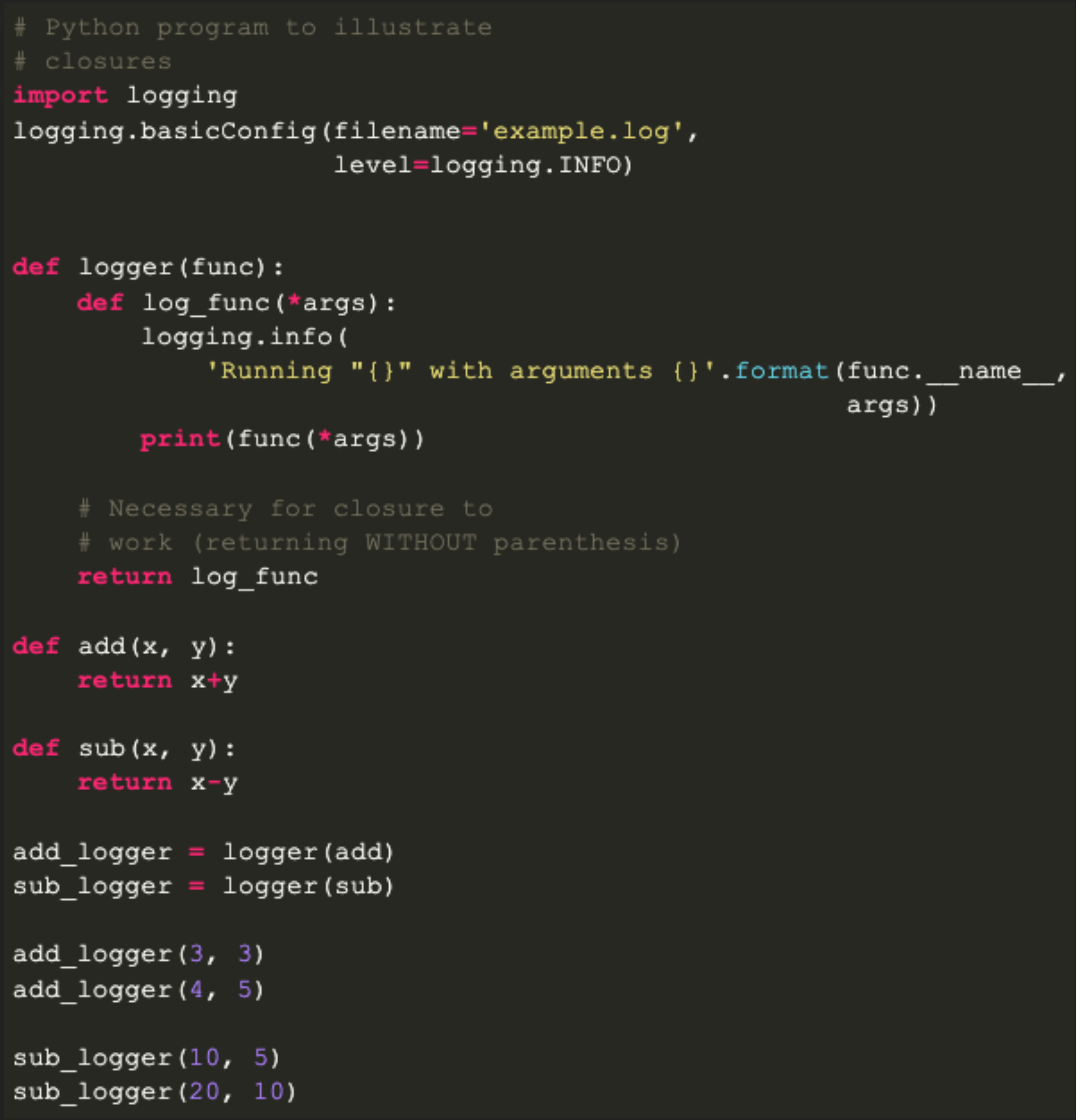
- in functional programming, closures make it possible to bind data to a function without actually passing the data as parameters
- this is similar to what a class does in oop

Closure example

- when you have a single method to be implemented in a class, closures can be used instead



```
def make_multiplier_of(n):
    def multiplier(x):
        return x * n
    return multiplier


# Multiplier of 3
times3 = make_multiplier_of(3)

# Multiplier of 5
times5 = make_multiplier_of(5)

# Output: 27
print(times3(9))

# Output: 15
print(times5(3))

# Output: 30
print(times5(times3(2)))
```



```python
# Python program to illustrate
# closures
import logging
logging.basicConfig(filename='example.log',
                    level=logging.INFO)


def logger(func):
    def log_func(*args):
        logging.info(
            'Running "{}" with arguments {}'.format(func.__name__,
                                                   args))
        print(func(*args))

    # Necessary for closure to
    # work (returning WITHOUT parenthesis)
    return log_func

def add(x, y):
    return x+y

def sub(x, y):
    return x-y

add_logger = logger(add)
sub_logger = logger(sub)

add_logger(3, 3)
add_logger(4, 5)

sub_logger(10, 5)
sub_logger(20, 10)
```

Closure example

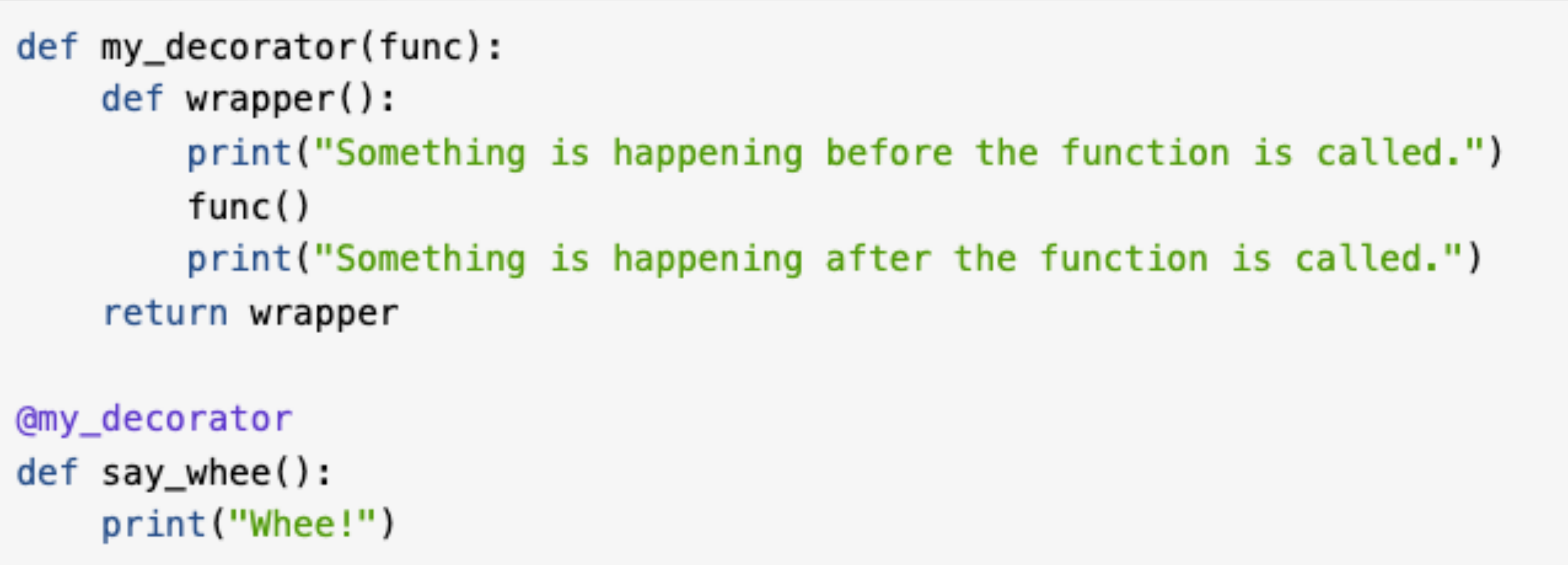
- output:
  - 6
  - 9
  - 5
  - 10
- closures can be used as callback functions
- closures can reduce the use of global variables

Decorators

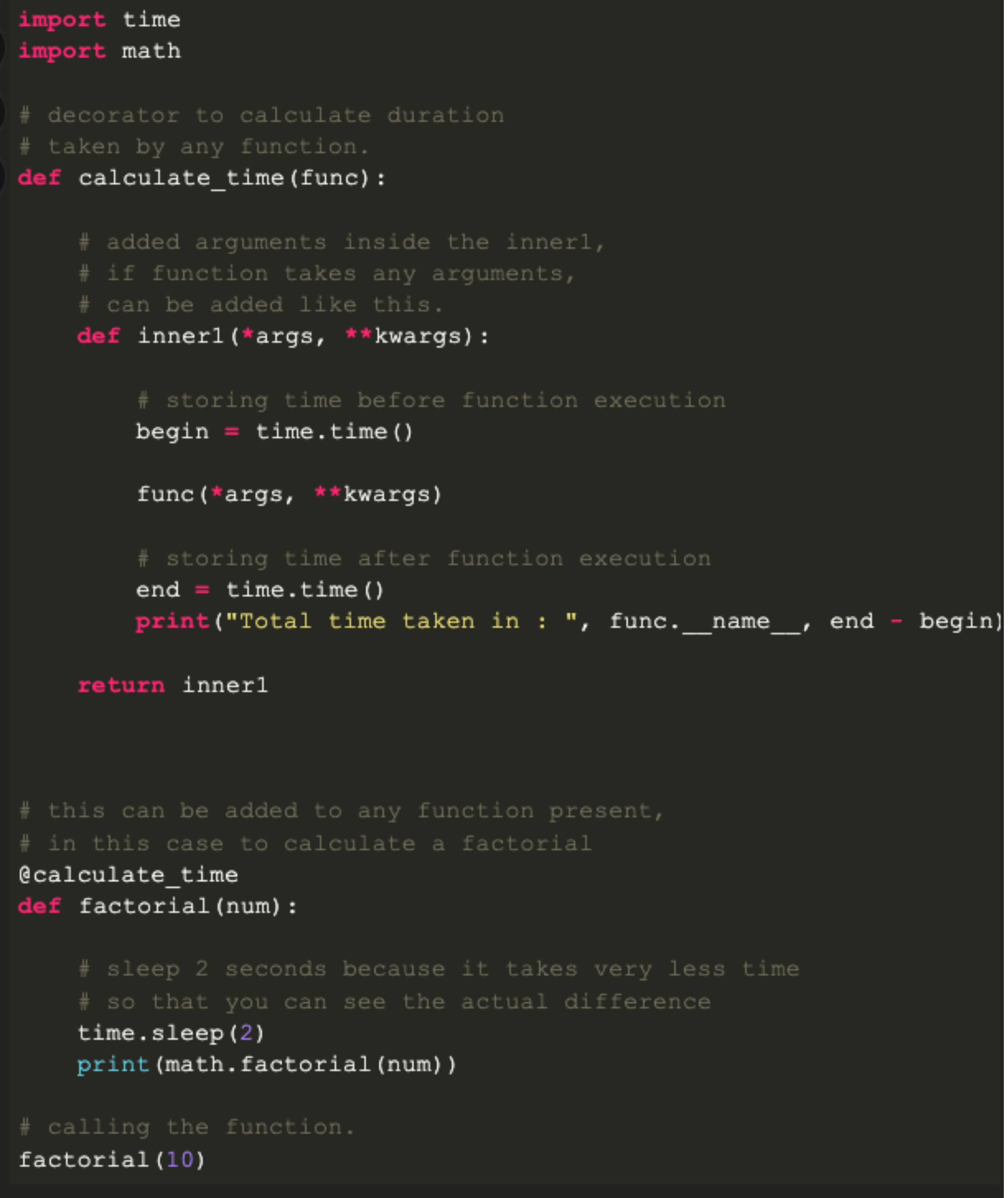
- decorators wrap a function, modifying its behavior

Decorator example

- the @my_decorator is a simpler way of saying say_whee = my_decorator(say_whee)



```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")
```



```python
import time
import math

# decorator to calculate duration
# taken by any function.
def calculate_time(func):

    # added arguments inside the inner1,
    # if function takes any arguments,
    # can be added like this.
    def inner1(*args, **kwargs):

        # storing time before function execution
        begin = time.time()

        func(*args, **kwargs)

        # storing time after function execution
        end = time.time()
        print("Total time taken in : ", func.__name__, end - begin)

    return inner1



# this can be added to any function present,
# in this case to calculate a factorial
@calculate_time
def factorial(num):

    # sleep 2 seconds because it takes very less time
    # so that you can see the actual difference
    time.sleep(2)
    print(math.factorial(num))

# calling the function.
factorial(10)
```

Decorator example

- find the execution time of a function using a decorator

Essential points to note

- WordNet is a lexical resource available through NLTK
- SentiWordNet and VADER can be used for sentiment analysis
- WSD word sense disambiguation tries to classify the particular sense of a word

To Do

- Portfolio: WordNet

Next topic

Ngrams