Introduction to Natural Language Processing using spaCy

Dr. Bambang Purnomosidi D. P.

1. Dean, Faculty of Information Technology, Universitas Teknologi Digital Indonesia (https://www.utdi.ac.id).
2. Director, Zimera Corporation (https://zimeracorp.com).

Agenda

- Computational Linguistics and NLP
- Introduction to spaCy
- Text Processing Pipelines
- Container Objects
- Linguistic Features of spaCy
- Similarity
- Sentence Pattern
- Intention
- Large Language Model

Computational Linguistics and NLP

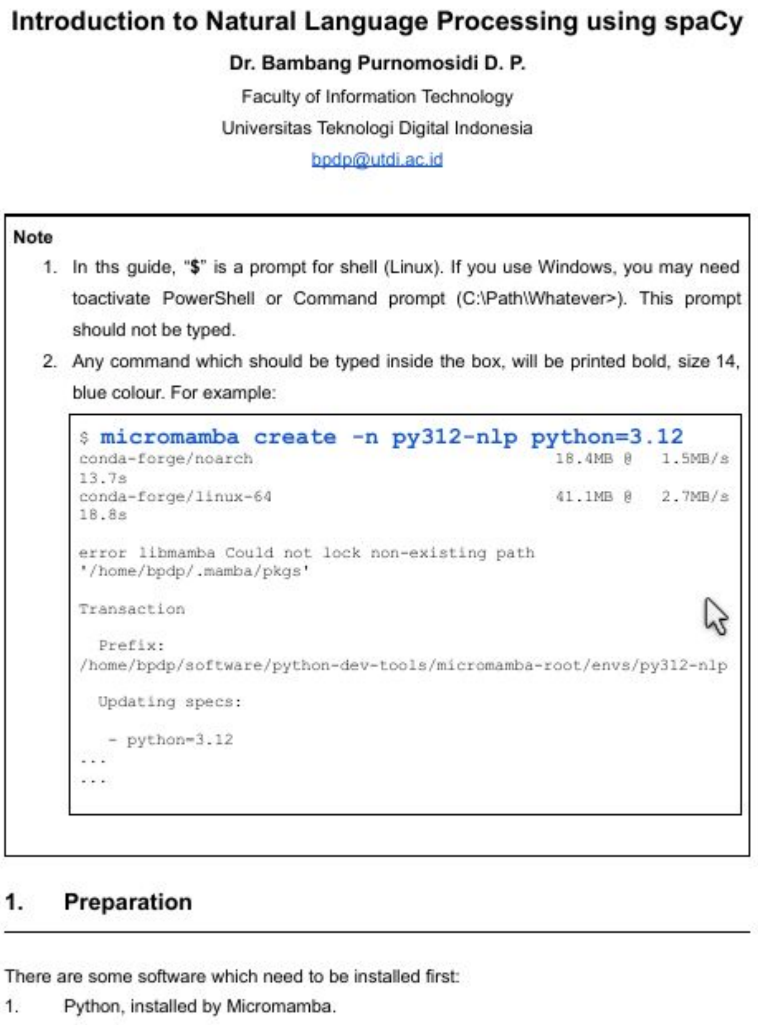
- **Natural Language**: a language developed by humans through natural use and communication, rather than through a process of deliberate construction and creation.
- To communicate using NL:
  - Understand the meaning of words, sentences, and grammar.
  - Understand the context: speech act theory.
  - Language cognition
  - Relation between language and reality
- **Computational Linguistics**: the scientific study of language from a computational perspective. Computational linguists are interested in providing computational models of various kinds of linguistic phenomena.
- **NLP**: the combination of Linguistics, AI, and Computer Science in relation to software engineering to process and analyze natural human language data

- CL is more research-oriented and has a broader scope than NLP. For example: modeling and identifying a language family.
- NLP is more engineering-oriented towards human natural language data and doing various things to produce certain outputs on the human natural language data. NLP uses various techniques and algorithms from CL to process human natural language data, while CL often uses NLP for the engineering side of the CL domain.
- Three sides of NLP:
    - **Speech Recognition**: recognize spoken expressions and translate these expressions into text.
    - **Natural Language Understanding**: understand the meaning contained in sentences or sentence units of natural human language.
    - **Natural Language Generation**: generate natural human language from data (linguistic and non-linguistic).

Introduction to spaCy

- From spaCy website (https://spacy.io):
  - spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It’s designed specifically for production use and helps you build applications that process and “understand” large volumes of text. It can be used to build information extraction or natural language understanding systems.
- spaCy is developed using Python (it needs Python <= 3.12 as of January 2025).
- From 3 sides of NLP, spaCy is used for NLU (Natural Language Understanding).
- How can one learn NLP using spaCy?

See the file (**workshop-spaCy preparation.pd**: for instructions on how to prepare your environment.



Introduction to Natural Language Processing using spaCy

Dr. Bambang Purnomosidi D. P.

Faculty of Information Technology

Universitas Teknologi Digital Indonesia

bpdp@utdi.ac.id

Note

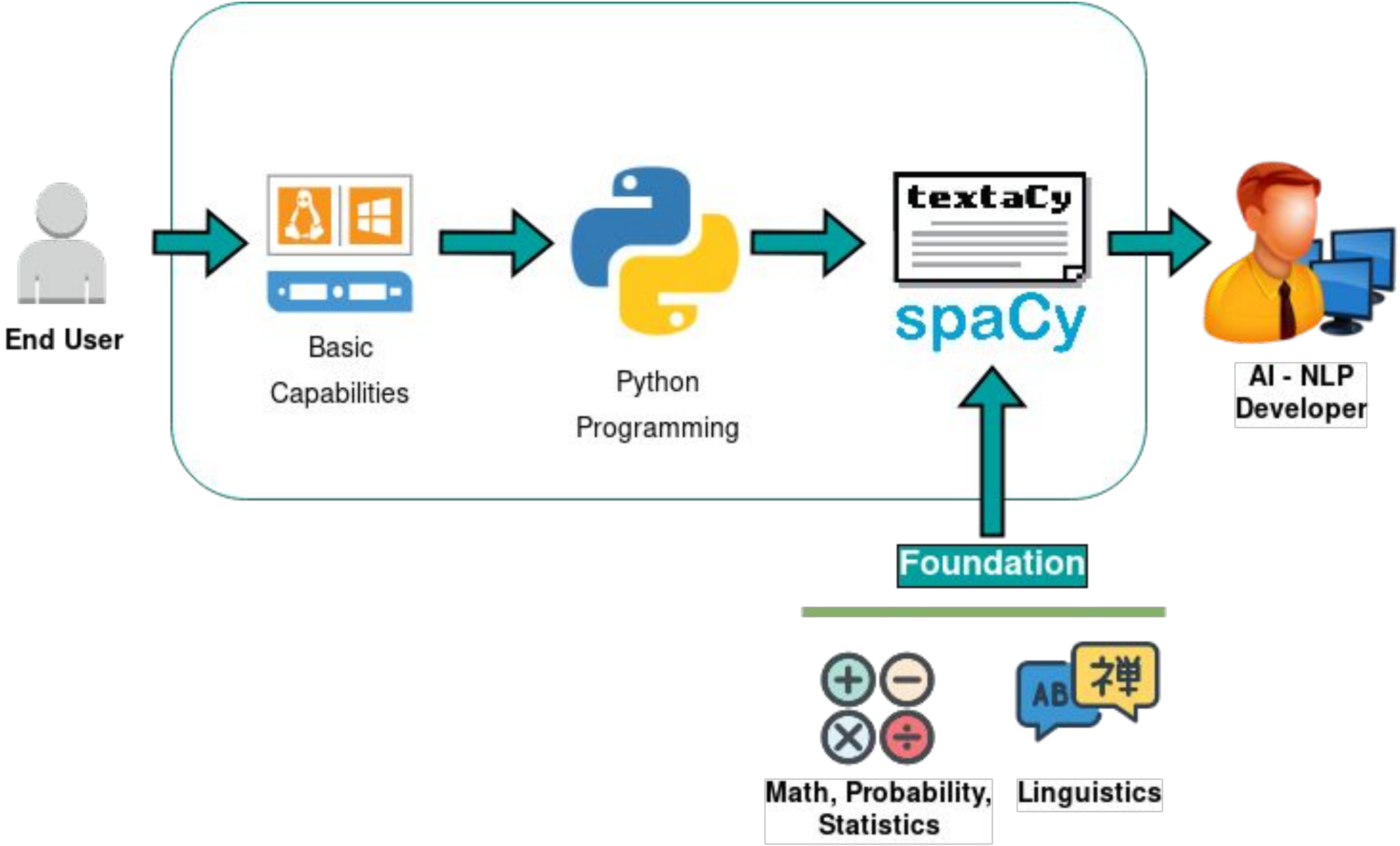
1. In ths guide, "**$**" is a prompt for shell (Linux). If you use Windows, you may need toactivate PowerShell or Command prompt (C:\Path\Whatever>). This prompt should not be typed.
2. Any command which should be typed inside the box, will be printed bold, size 14, blue colour. For example:

```
$ micromamba create -n py312-nlp python=3.12
conda-forge/noarch                                   18.4MB @   1.5MB/s
13.7s
conda-forge/linux-64                                 41.1MB @   2.7MB/s
18.8s

error libmamba Could not lock non-existing path
'/home/bpdp/.mamba/pkgs'

Transaction

  Prefix:
/home/bpdp/software/python-dev-tools/micromamba-root/envs/py312-nlp

  Updating specs:

   - python=3.12
...
...
```

1. Preparation

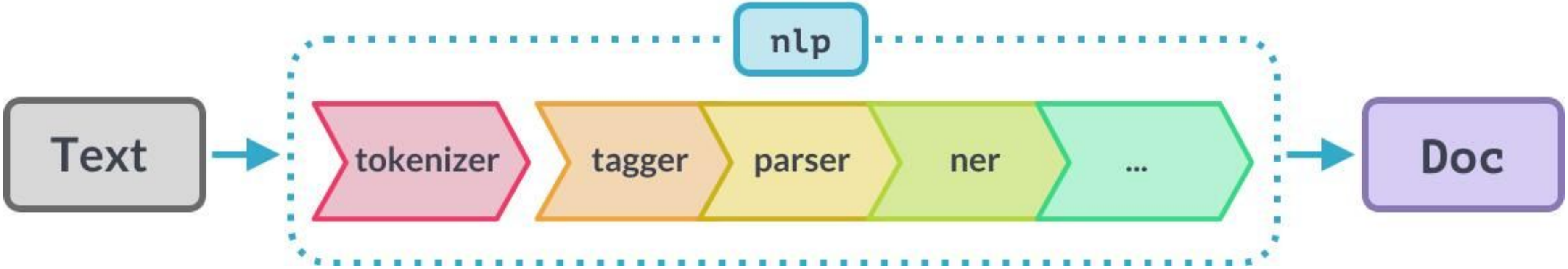
There are some software which need to be installed first:

1. Python, installed by Micromamba.

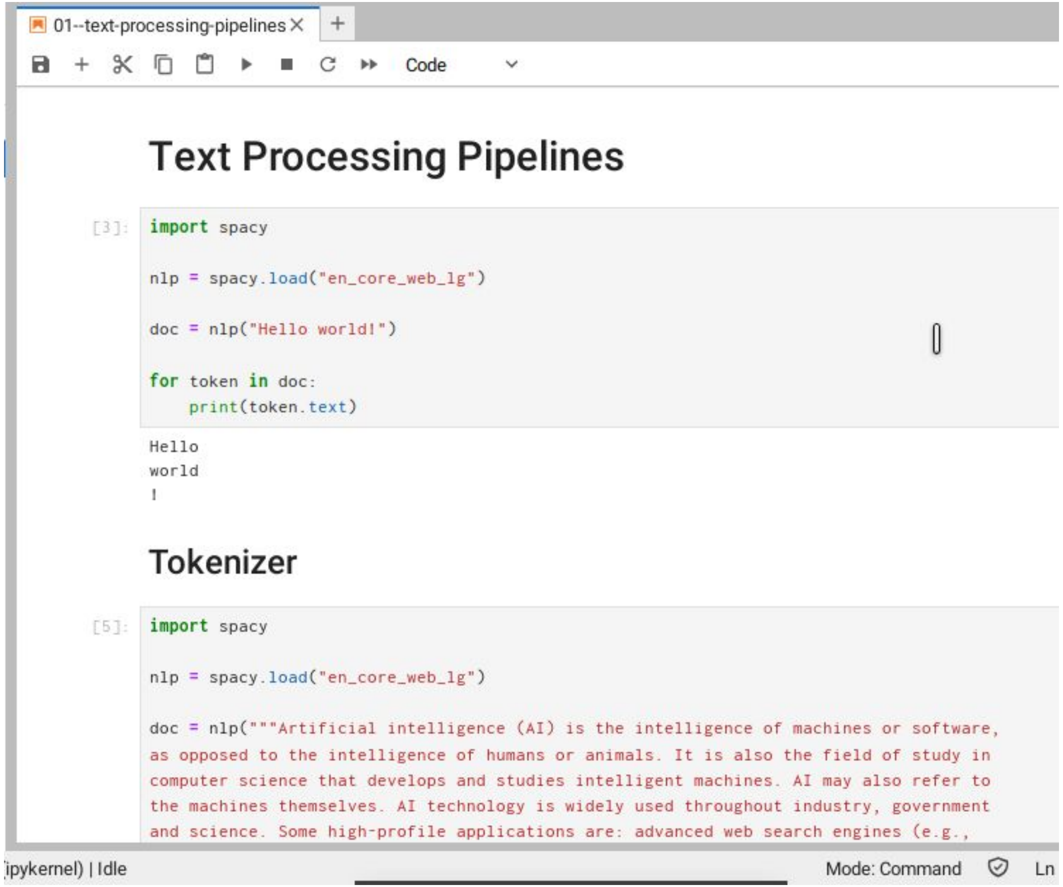


End User
Basic Capabilities
Python Programming
textaCy
spaCy
AI - NLP Developer
Foundation
Math, Probability, Statistics
Linguistics
AB
禅

Text Processing Pipelines



1. ***Tokenizer***: segmenting text into words, punctuation and the like.
2. ***Tagger***: predicts PoS (parts of speech) tag.
3. ***Parser*** or ***dependency parser***: describe the syntactic dependency between 2 words in a sentence.
4. ***NER***: named entity recognition.



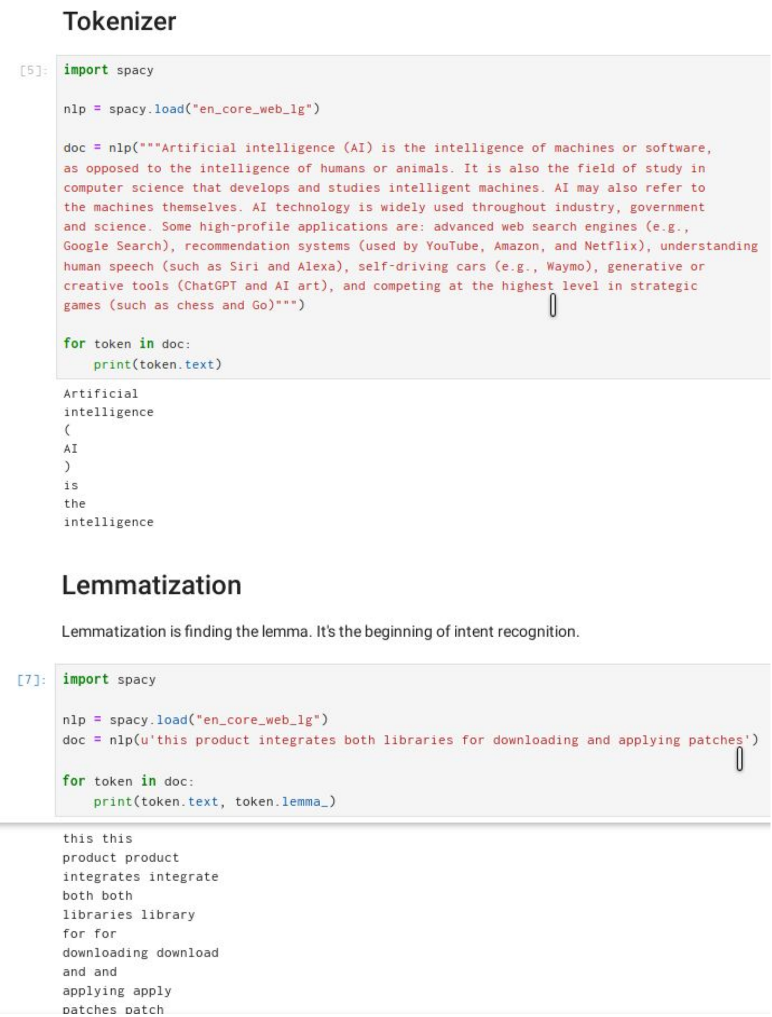
Text Processing Pipelines

```
[3]: import spacy

     nlp = spacy.load("en_core_web_lg")

     doc = nlp("Hello world!")

     for token in doc:
         print(token.text)
```

```
Hello
world
!
```

Tokenizer

```
[5]: import spacy

     nlp = spacy.load("en_core_web_lg")

     doc = nlp("""Artificial intelligence (AI) is the intelligence of machines or software,
     as opposed to the intelligence of humans or animals. It is also the field of study in
     computer science that develops and studies intelligent machines. AI may also refer to
     the machines themselves. AI technology is widely used throughout industry, government
     and science. Some high-profile applications are: advanced web search engines (e.g.,
```



Tokenizer

```
[5]: import spacy

     nlp = spacy.load("en_core_web_lg")

     doc = nlp("""Artificial intelligence (AI) is the intelligence of machines or software,
     as opposed to the intelligence of humans or animals. It is also the field of study in
     computer science that develops and studies intelligent machines. AI may also refer to
     the machines themselves. AI technology is widely used throughout industry, government
     and science. Some high-profile applications are: advanced web search engines (e.g.,
     Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding
     human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or
     creative tools (ChatGPT and AI art), and competing at the highest level in strategic
     games (such as chess and Go)""")

     for token in doc:
         print(token.text)
```

```
Artificial
intelligence
(
AI
)
is
the
intelligence
```

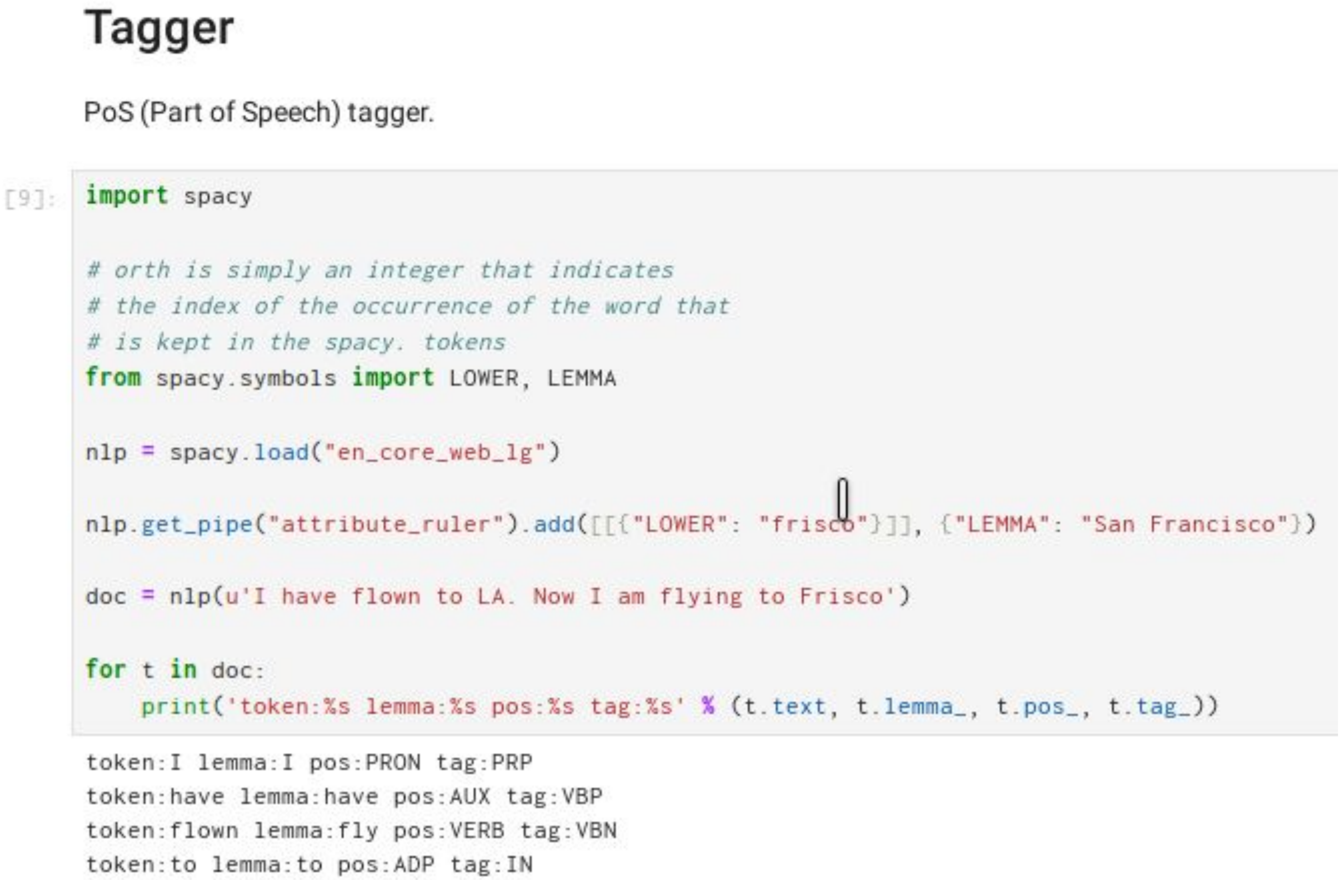
Lemmatization

Lemmatization is finding the lemma. It's the beginning of intent recognition.

```
[7]: import spacy

     nlp = spacy.load("en_core_web_lg")
     doc = nlp(u'this product integrates both libraries for downloading and applying patches')

     for token in doc:
         print(token.text, token.lemma_)
```

```
this this
product product
integrates integrate
both both
libraries library
for for
downloading download
and and
applying apply
patches patch
```

See ***01--text-processing-pipelines.ipynb***



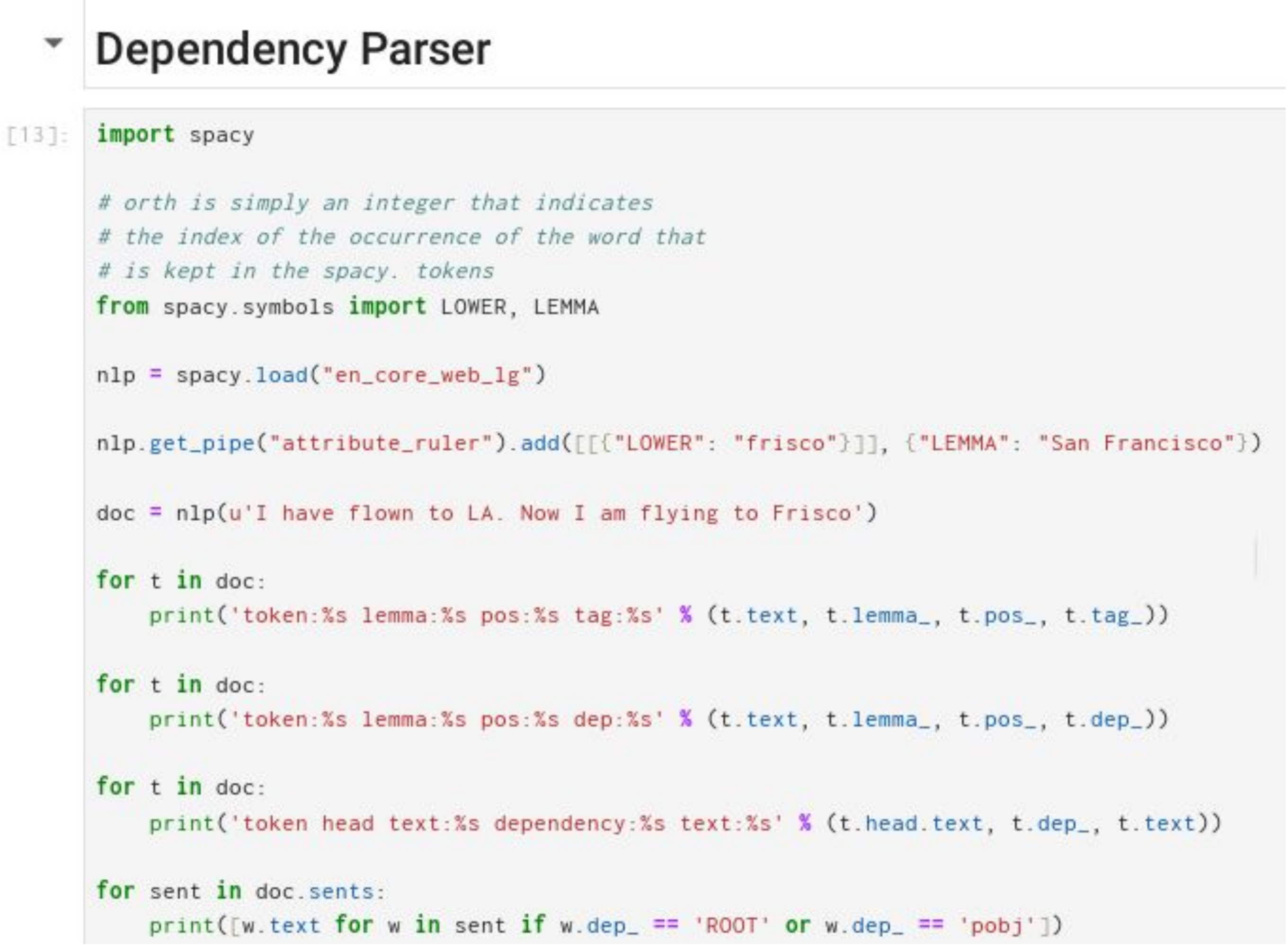
Tagger

PoS (Part of Speech) tagger.

[9]:

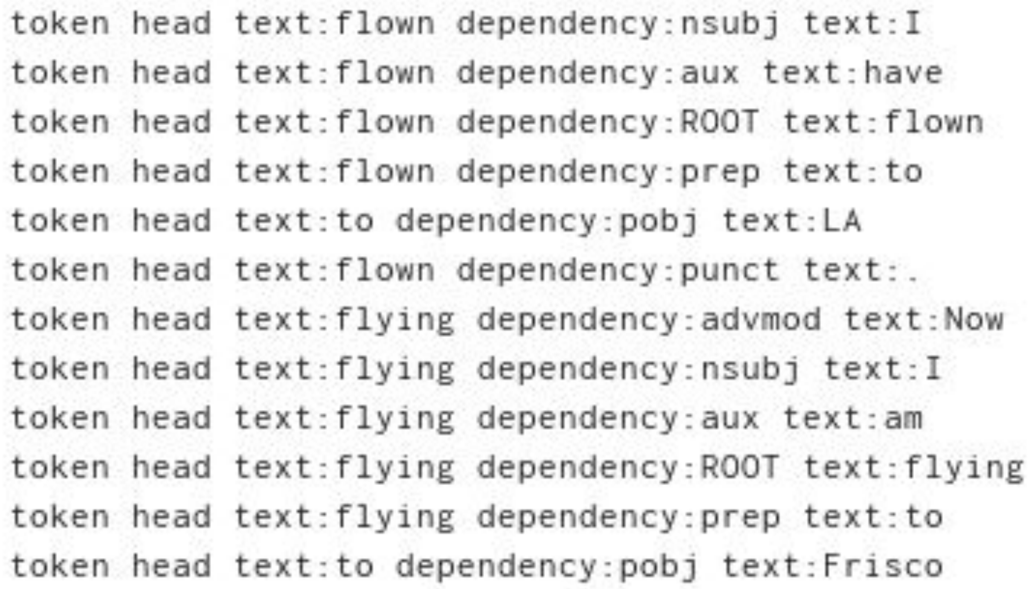
```
import spacy

# orth is simply an integer that indicates
# the index of the occurrence of the word that
# is kept in the spacy. tokens
from spacy.symbols import LOWER, LEMMA

nlp = spacy.load("en_core_web_lg")

nlp.get_pipe("attribute_ruler").add([[{"LOWER": "frisco"}]], {"LEMMA": "San Francisco"})

doc = nlp(u'I have flown to LA. Now I am flying to Frisco')

for t in doc:
    print('token:%s lemma:%s pos:%s tag:%s' % (t.text, t.lemma_, t.pos_, t.tag_))
```

```
token:I lemma:I pos:PRON tag:PRP
token:have lemma:have pos:AUX tag:VBP
token:flown lemma:fly pos:VERB tag:VBN
token:to lemma:to pos:ADP tag:IN
```



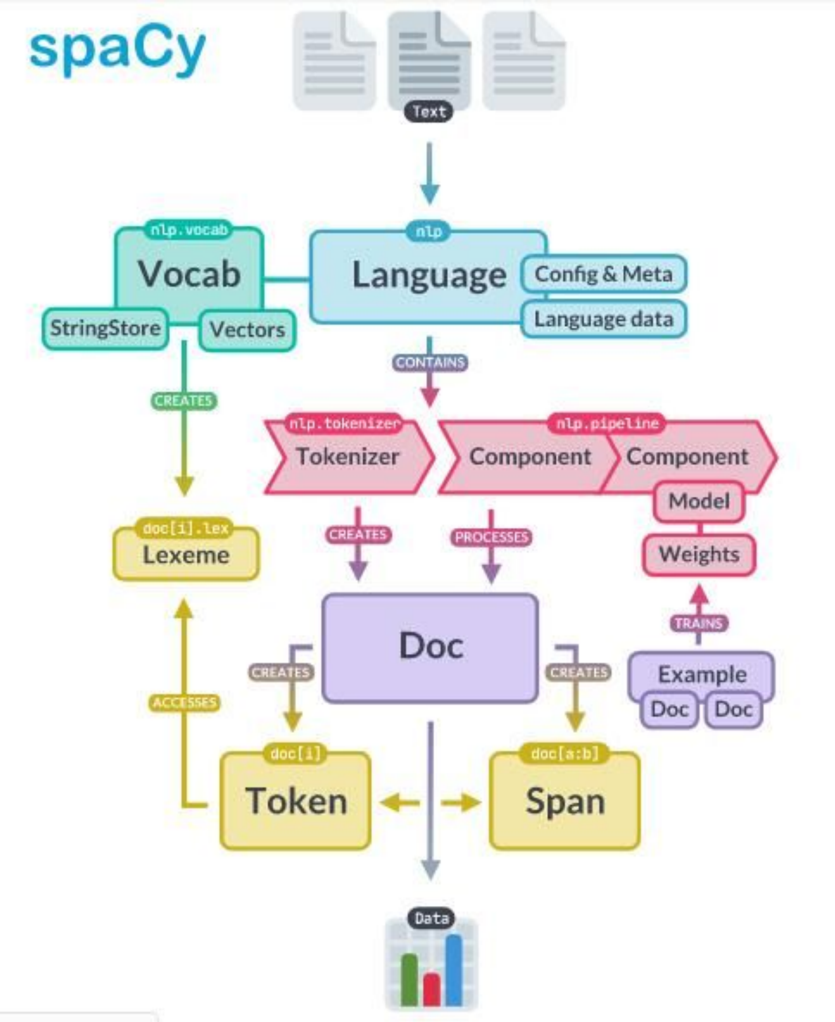
Dependency Parser

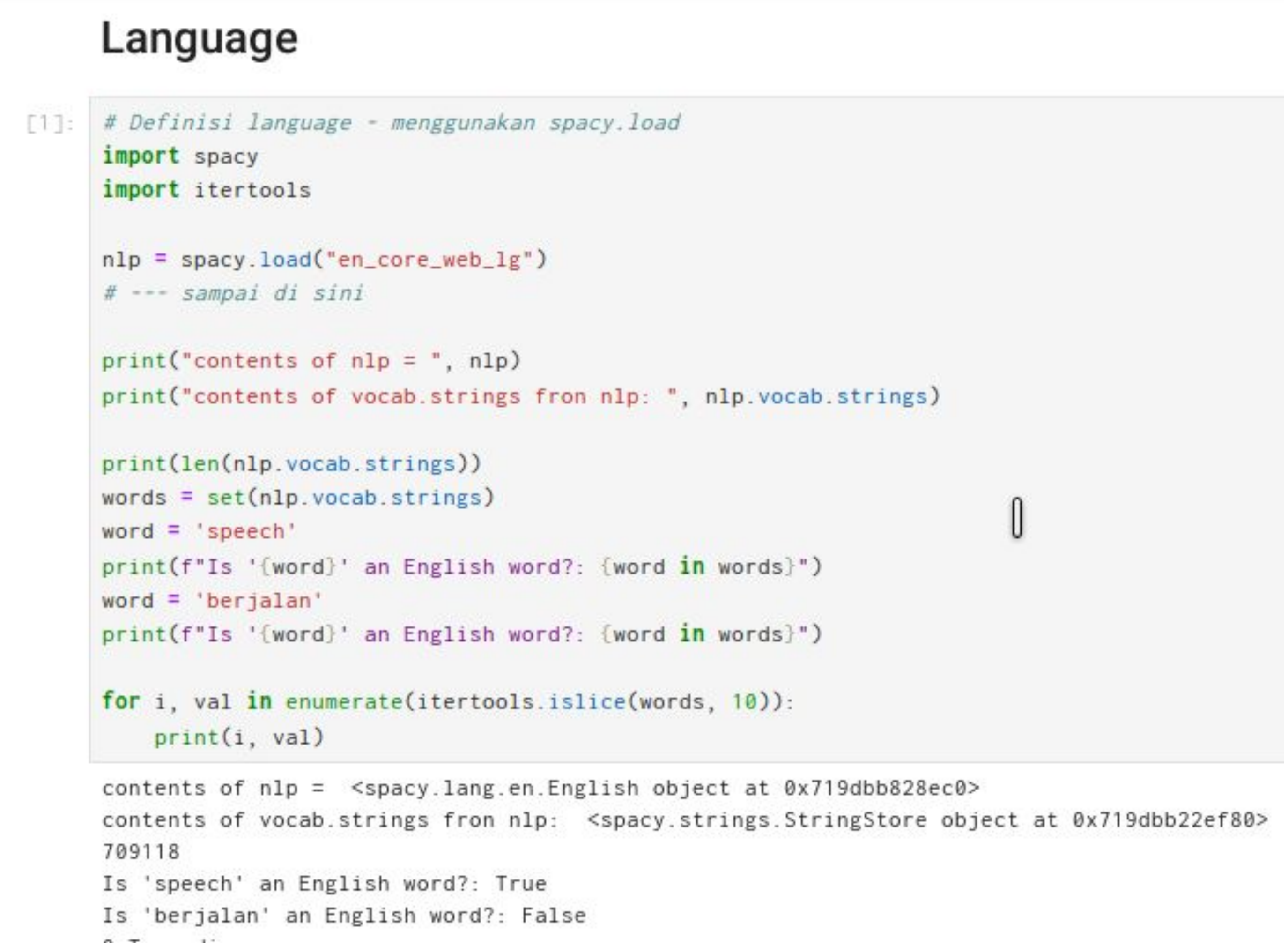
```
import spacy

# orth is simply an integer that indicates
# the index of the occurrence of the word that
# is kept in the spacy. tokens
from spacy.symbols import LOWER, LEMMA

nlp = spacy.load("en_core_web_lg")

nlp.get_pipe("attribute_ruler").add([[{"LOWER": "frisco"}]], {"LEMMA": "San Francisco"})

doc = nlp(u'I have flown to LA. Now I am flying to Frisco')

for t in doc:
    print('token:%s lemma:%s pos:%s tag:%s' % (t.text, t.lemma_, t.pos_, t.tag_))

for t in doc:
    print('token:%s lemma:%s pos:%s dep:%s' % (t.text, t.lemma_, t.pos_, t.dep_))

for t in doc:
    print('token head text:%s dependency:%s text:%s' % (t.head.text, t.dep_, t.text))

for sent in doc.sents:
    print([w.text for w in sent if w.dep_ == 'ROOT' or w.dep_ == 'pobj'])
```



```
token head text:flown dependency:nsubj text:I
token head text:flown dependency:aux text:have
token head text:flown dependency:ROOT text:flown
token head text:flown dependency:prep text:to
token head text:to dependency:pobj text:LA
token head text:flown dependency:punct text:.
token head text:flying dependency:advmod text:Now
token head text:flying dependency:nsubj text:I
token head text:flying dependency:aux text:am
token head text:flying dependency:ROOT text:flying
token head text:flying dependency:prep text:to
token head text:to dependency:pobj text:Frisco
```

Container Objects



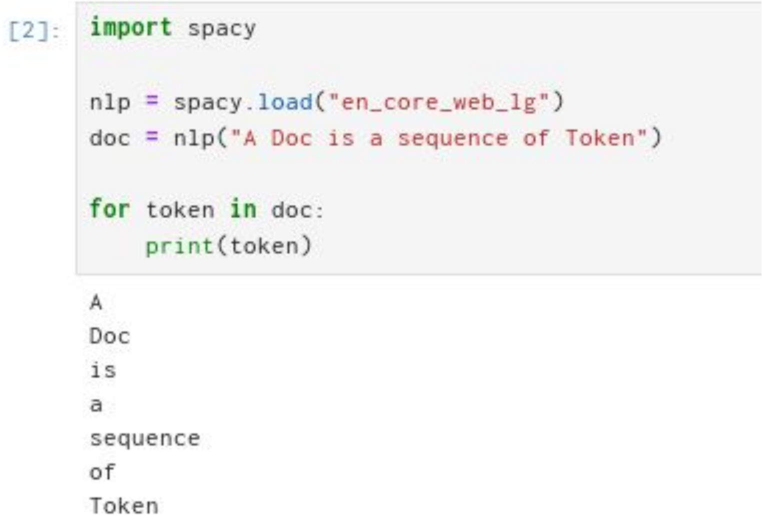


Language

```
# Definisi language - menggunakan spacy.load
import spacy
import itertools

nlp = spacy.load("en_core_web_lg")
# --- sampai di sini

print("contents of nlp = ", nlp)
print("contents of vocab.strings fron nlp: ", nlp.vocab.strings)

print(len(nlp.vocab.strings))
words = set(nlp.vocab.strings)
word = 'speech'
print(f"Is '{word}' an English word?: {word in words}")
word = 'berjalan'
print(f"Is '{word}' an English word?: {word in words}")

for i, val in enumerate(itertools.islice(words, 10)):
    print(i, val)
```

```
contents of nlp =  <spacy.lang.en.English object at 0x719dbb828ec0>
contents of vocab.strings fron nlp:  <spacy.strings.StringStore object at 0x719dbb22ef80>
709118
Is 'speech' an English word?: True
Is 'berjalan' an English word?: False
```

Doc



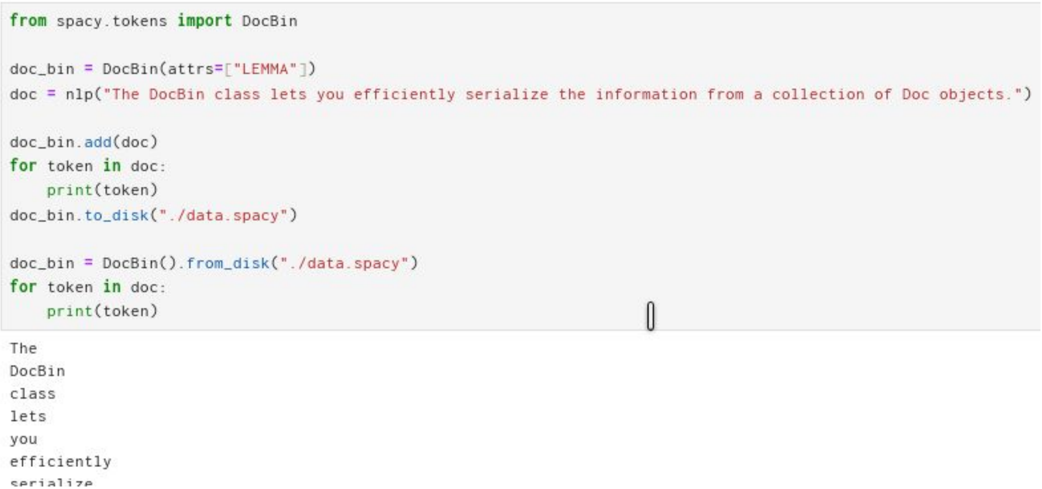
[2]:

```
import spacy

nlp = spacy.load("en_core_web_lg")
doc = nlp("A Doc is a sequence of Token")

for token in doc:
    print(token)
```

```
A
Doc
is
a
sequence
of
Token
```

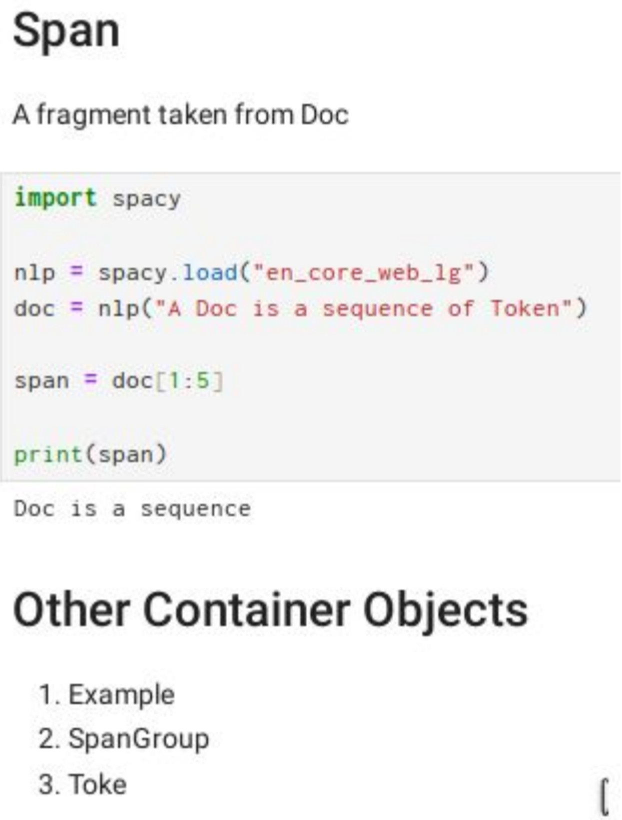
DocBin

If you need to serialize processing results into a binary object.



```
from spacy.tokens import DocBin

doc_bin = DocBin(attrs=["LEMMA"])
doc = nlp("The DocBin class lets you efficiently serialize the information from a collection of Doc objects.")

doc_bin.add(doc)
for token in doc:
    print(token)
doc_bin.to_disk("./data.spacy")

doc_bin = DocBin().from_disk("./data.spacy")
for token in doc:
    print(token)
```

```
The
DocBin
class
lets
you
efficiently
serialize
```



Span

A fragment taken from Doc

```python
import spacy

nlp = spacy.load("en_core_web_lg")
doc = nlp("A Doc is a sequence of Token")

span = doc[1:5]

print(span)
```

```
Doc is a sequence
```

Other Container Objects

1. Example
2. SpanGroup
3. Toke

Linguistics Features of spaCy

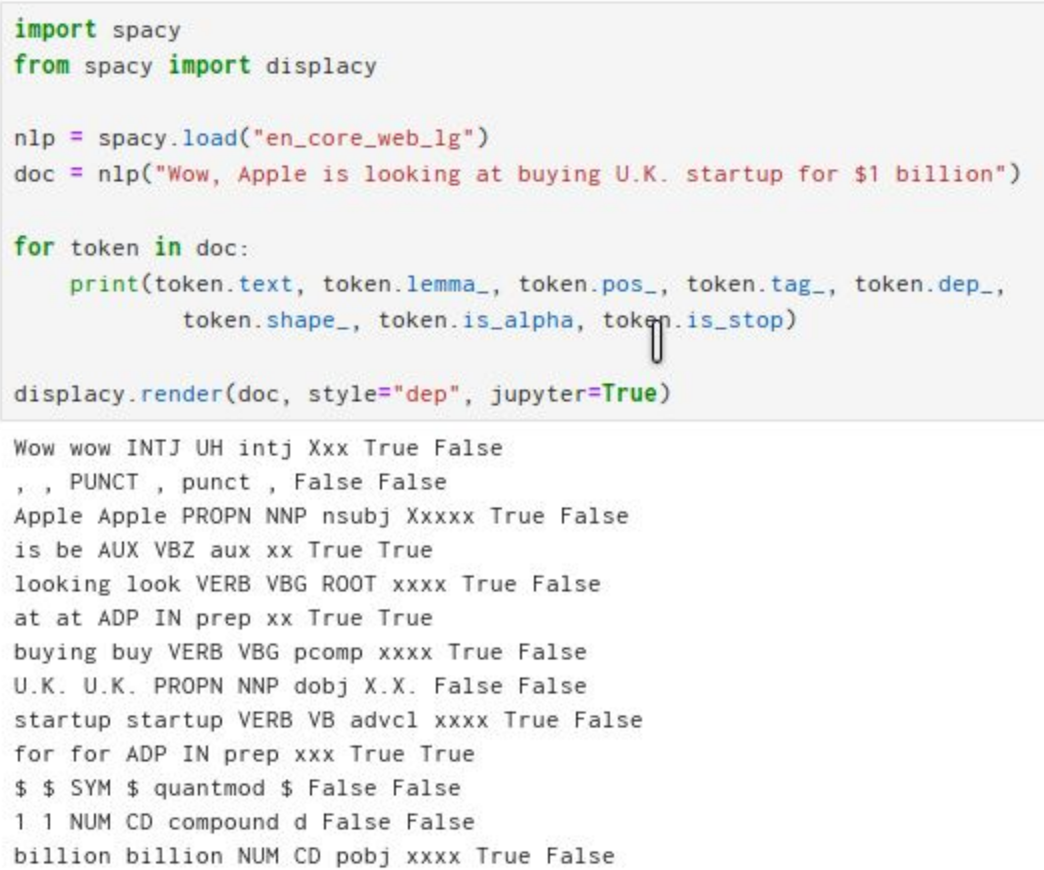
- Processing raw text data is not easy.
- Processing the data raw character by character and word by word is indeed quite possible to be a solution to simple problems (e.g.: how many times a word appears, etc.), but it would be better if linguistic knowledge was applied to the raw text.
- Raw text => spaCy => Doc
- The Doc object has a set of linguistic annotations.

PoS (Part-of-Speech) Tagging

- *Part-of-speech*: a category of words or lexical items that have similar grammatical properties.
- *lexical item*: a word, part of a word, or group of words that form the basic elements of a lexicon (vocabulary). Examples of lexical items: by the way, it's raining cats and dogs, -able, -er, good
- POS tagging (also known as PoS tagging or POST): the assignment of a special label given to a token (word) in a text corpus based on definition and context.

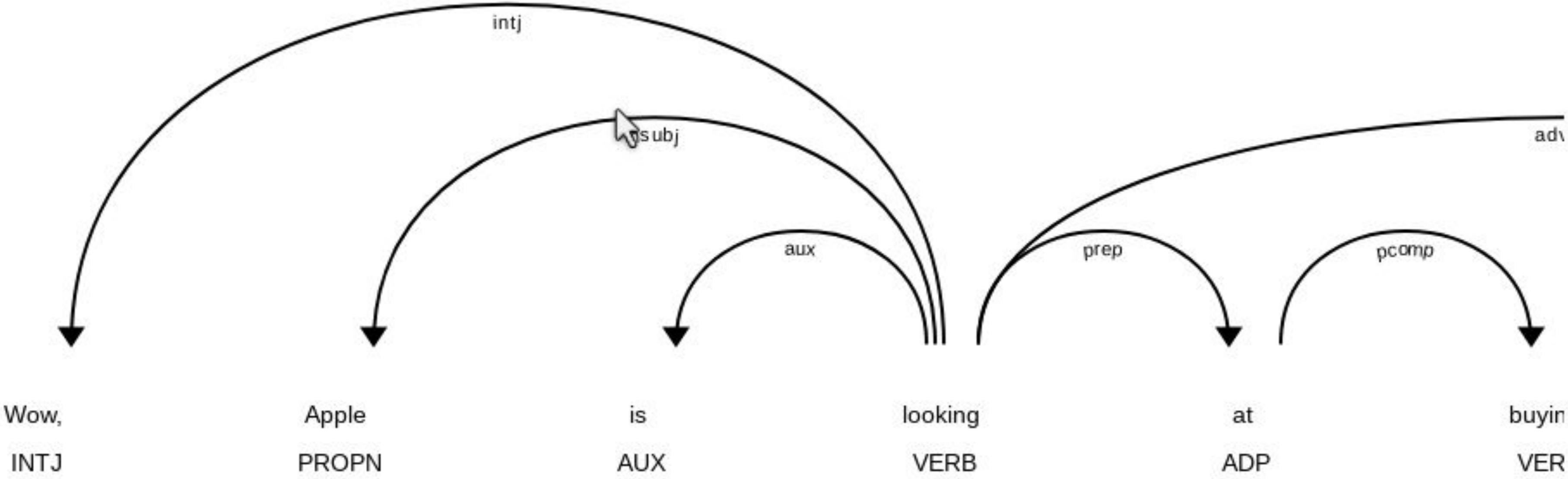
Some PoS:

1. noun (NOUN)
2. verb (VERB).
3. determiner (DET)
4. adjective (ADJ)
5. adposition (ADP) preposition (on, of, at, with, ...) and postposition (... ago).
6. pronoun (PRON)
7. adverb (ADV)
8. conjunction (CONJ
9. interjection (INTJ)



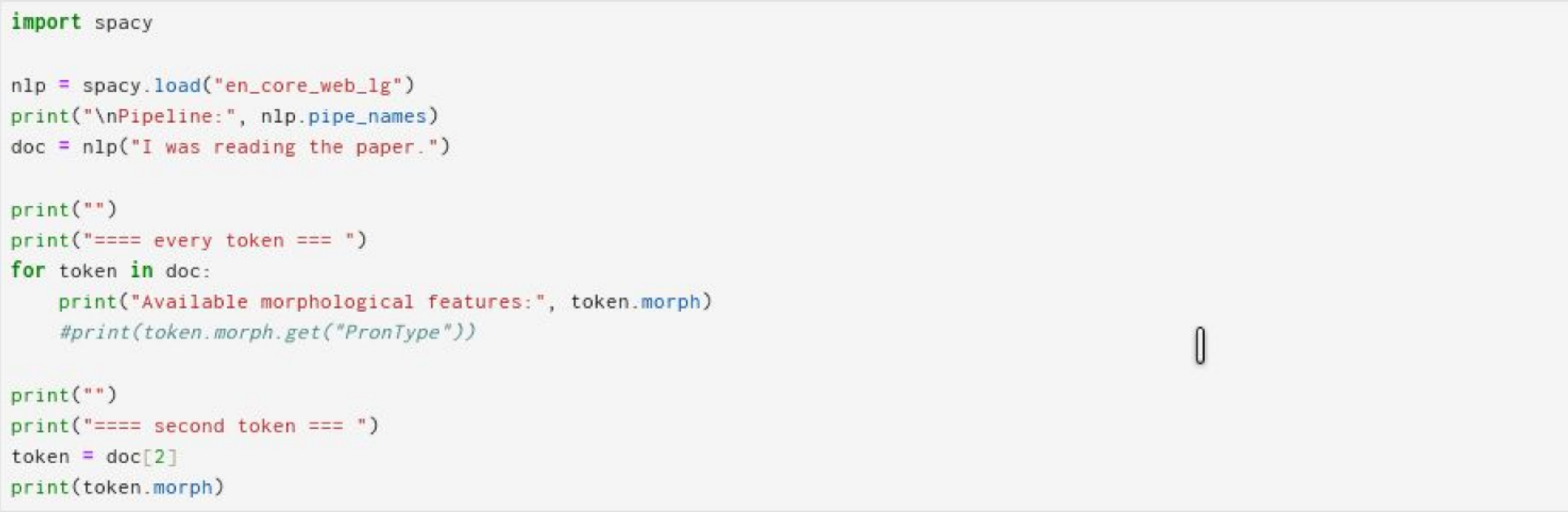
```python
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_lg")
doc = nlp("Wow, Apple is looking at buying U.K. startup for $1 billion")

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
            token.shape_, token.is_alpha, token.is_stop)

displacy.render(doc, style="dep", jupyter=True)
```

```
Wow wow INTJ UH intj Xxx True False
, , PUNCT , punct , False False
Apple Apple PROPN NNP nsubj Xxxxx True False
is be AUX VBZ aux xx True True
looking look VERB VBG ROOT xxxx True False
at at ADP IN prep xx True True
buying buy VERB VBG pcomp xxxx True False
U.K. U.K. PROPN NNP dobj X.X. False False
startup startup VERB VB advcl xxxx True False
for for ADP IN prep xxx True True
$ $ SYM $ quantmod $ False False
1 1 NUM CD compound d False False
billion billion NUM CD pobj xxxx True False
```

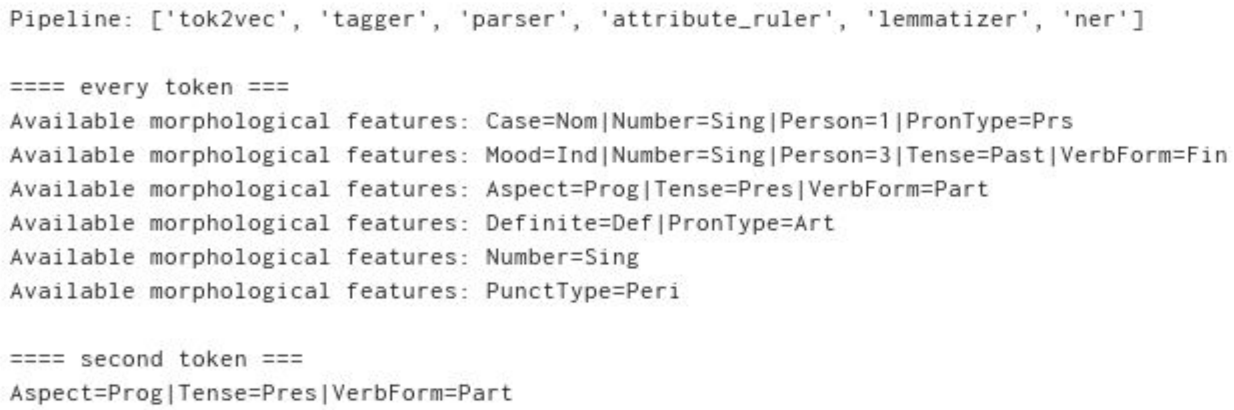

intj
nsubj
aux
prep
pcomp
adv
Wow,
INTJ
Apple
PROPN
is
AUX
looking
VERB
at
ADP
buyin
VER

Morphology

Morphology analyzes the structure of words and the effects of changes in structure on meaning and word class. For example, in English, the addition of -ing can be given to gerunds. The basic form of a word is modified by adding prefixes or suffixes that determine its grammatical function but do not change its part-of-speech.



```
import spacy

nlp = spacy.load("en_core_web_lg")
print("\nPipeline:", nlp.pipe_names)
doc = nlp("I was reading the paper.")

print("")
print("==== every token === ")
for token in doc:
    print("Available morphological features:", token.morph)
    #print(token.morph.get("PronType"))

print("")
print("==== second token === ")
token = doc[2]
print(token.morph)
```

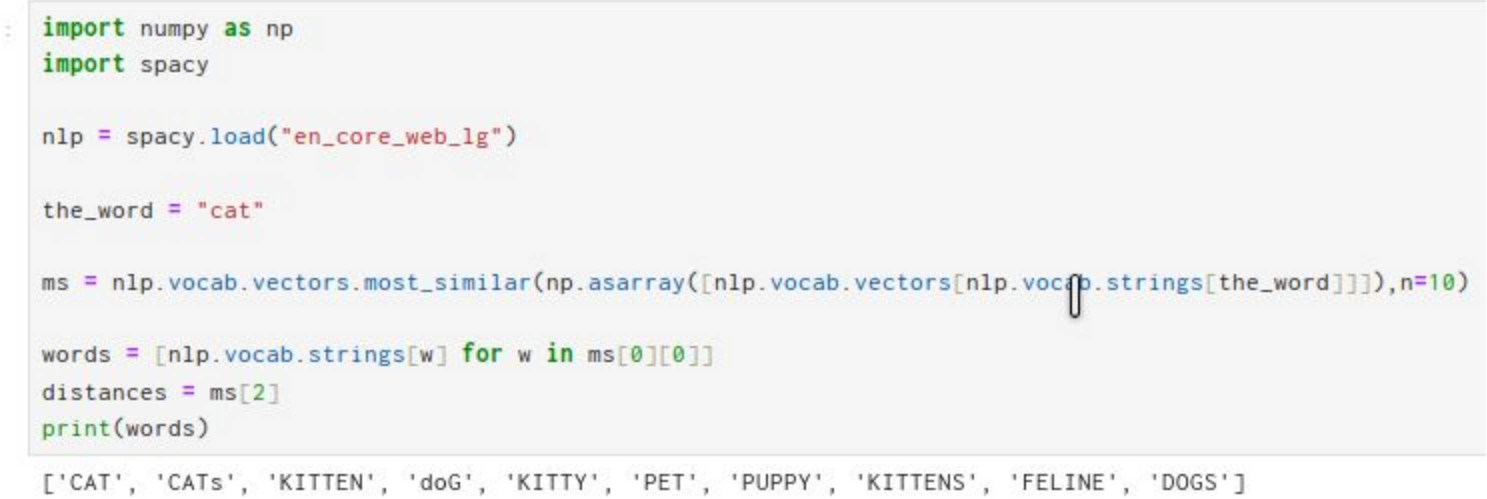


```
Pipeline: ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']

==== every token ===
Available morphological features: Case=Nom|Number=Sing|Person=1|PronType=Prs
Available morphological features: Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin
Available morphological features: Aspect=Prog|Tense=Pres|VerbForm=Part
Available morphological features: Definite=Def|PronType=Art
Available morphological features: Number=Sing
Available morphological features: PunctType=Peri

==== second token ===
Aspect=Prog|Tense=Pres|VerbForm=Part
```

Similarity

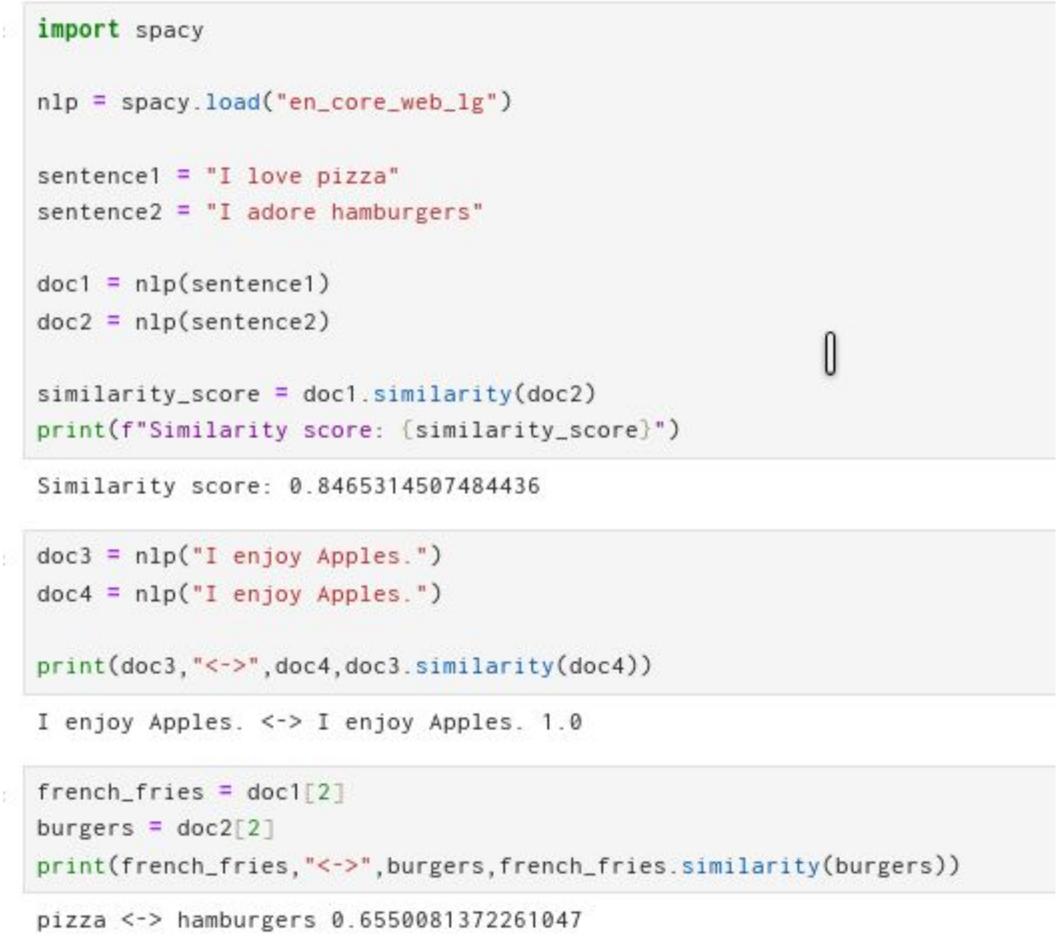
- Measuring the similarity between sentences / words / documents
- ***Computing Token Similarity***: Token similarity refers to measuring the similarity between individual words or tokens in a text based on their linguistic characteristics.
- spaCy provides pre-trained word vectors, Spacy can calculate the similarity score between two tokens using cosine similarity or other distance metrics.
- ***Word Vectors*** are numerical representations of words in multidimensional space through matrices. WV sometimes also called as word embeddings. The purpose of the word vector is to get a computer system to understand a word

Similarity



```python
import numpy as np
import spacy

nlp = spacy.load("en_core_web_lg")

the_word = "cat"

ms = nlp.vocab.vectors.most_similar(np.asarray([nlp.vocab.vectors[nlp.vocab.strings[the_word]]]),n=10)

words = [nlp.vocab.strings[w] for w in ms[0][0]]
distances = ms[2]
print(words)
```

```
['CAT', 'CATs', 'KITTEN', 'doG', 'KITTY', 'PET', 'PUPPY', 'KITTENS', 'FELINE', 'DOGS']
```



```
import spacy

nlp = spacy.load("en_core_web_lg")

sentence1 = "I love pizza"
sentence2 = "I adore hamburgers"

doc1 = nlp(sentence1)
doc2 = nlp(sentence2)

similarity_score = doc1.similarity(doc2)
print(f"Similarity score: {similarity_score}")
```

```
Similarity score: 0.8465314507484436
```

```
doc3 = nlp("I enjoy Apples.")
doc4 = nlp("I enjoy Apples.")

print(doc3,"<->",doc4,doc3.similarity(doc4))
```

```
I enjoy Apples. <-> I enjoy Apples. 1.0
```

```
french_fries = doc1[2]
burgers = doc2[2]
print(french_fries,"<->",burgers,french_fries.similarity(burgers))
```

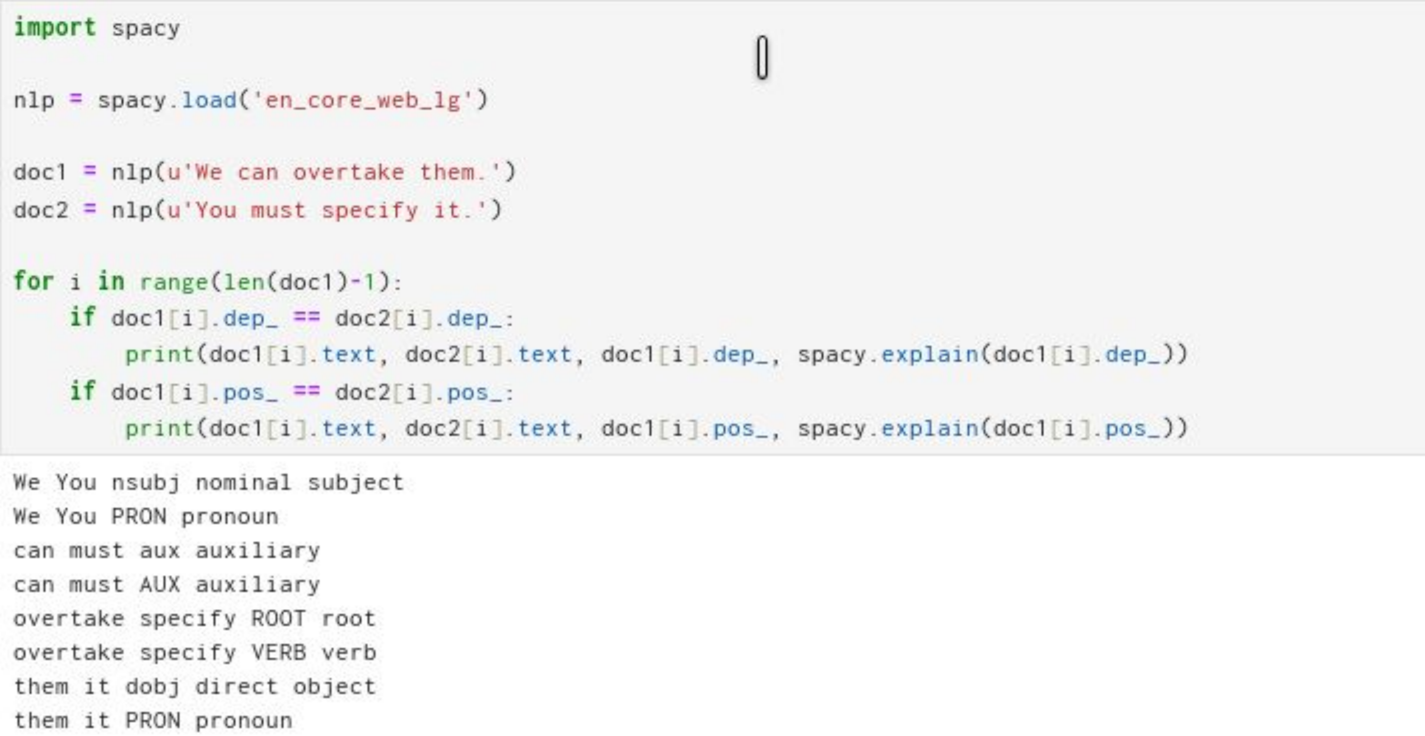
```
pizza <-> hamburgers 0.6550081372261047
```

Sentence Pattern

- In conducting analysis to understand sentences, we can use word sequence patterns.
- Its main function is usually used for classification (for example, we can classify a sentence as an interrogative sentence asking about ability) and generating text.
- In addition, there is a technique walking the syntactic dependency tree to retrieve certain information in a sentence.

Sentence Pattern

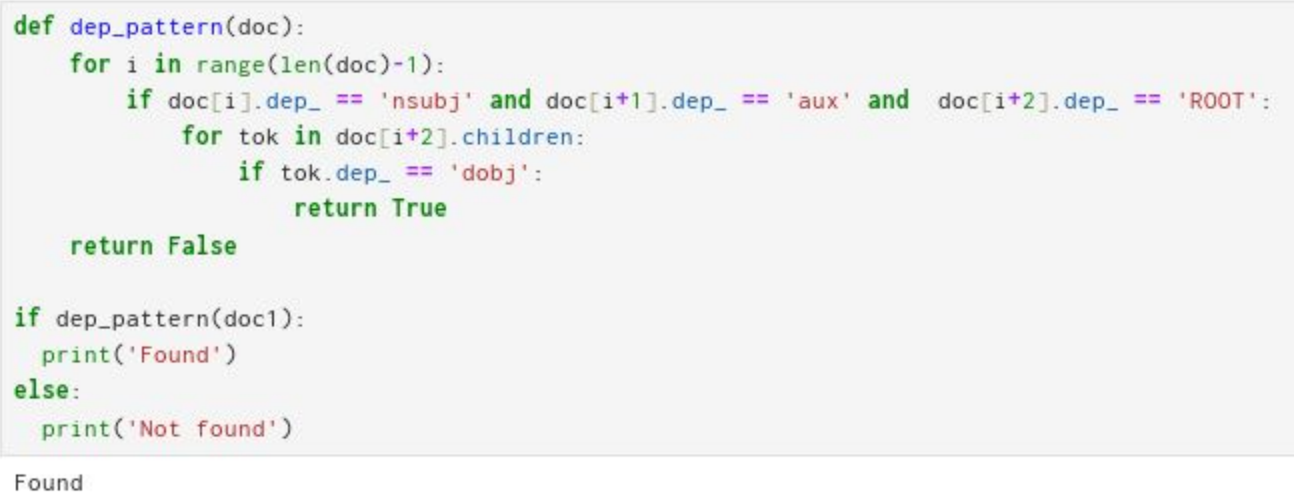
Note: source code was taken from https://nostarch.com/NLPPython but adjusted for latext spaCy version (3.8.x)



```python
import spacy

nlp = spacy.load('en_core_web_lg')

doc1 = nlp(u'We can overtake them.')
doc2 = nlp(u'You must specify it.')

for i in range(len(doc1)-1):
    if doc1[i].dep_ == doc2[i].dep_:
        print(doc1[i].text, doc2[i].text, doc1[i].dep_, spacy.explain(doc1[i].dep_))
    if doc1[i].pos_ == doc2[i].pos_:
        print(doc1[i].text, doc2[i].text, doc1[i].pos_, spacy.explain(doc1[i].pos_))
```

```
We You nsubj nominal subject
We You PRON pronoun
can must aux auxiliary
can must AUX auxiliary
overtake specify ROOT root
overtake specify VERB verb
them it dobj direct object
them it PRON pronoun
```

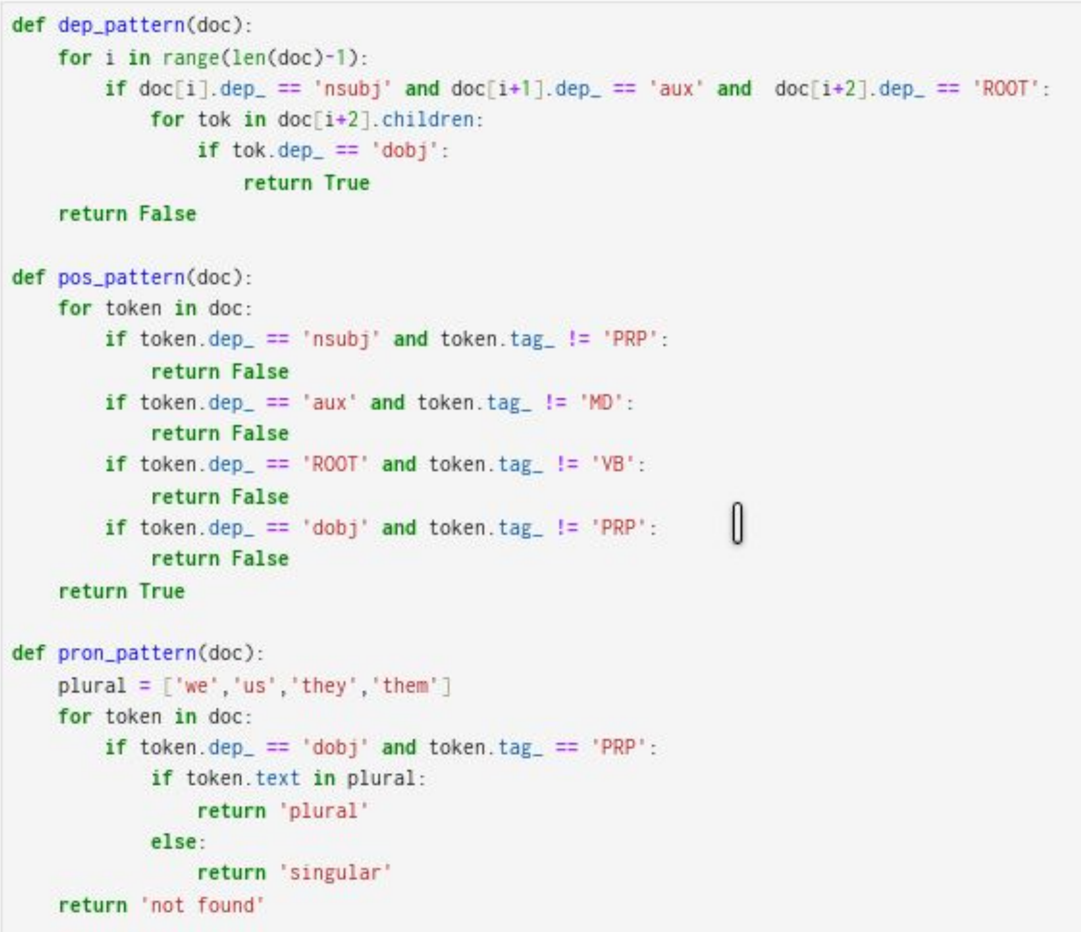
We may check whether a sentence has a specific pattern:



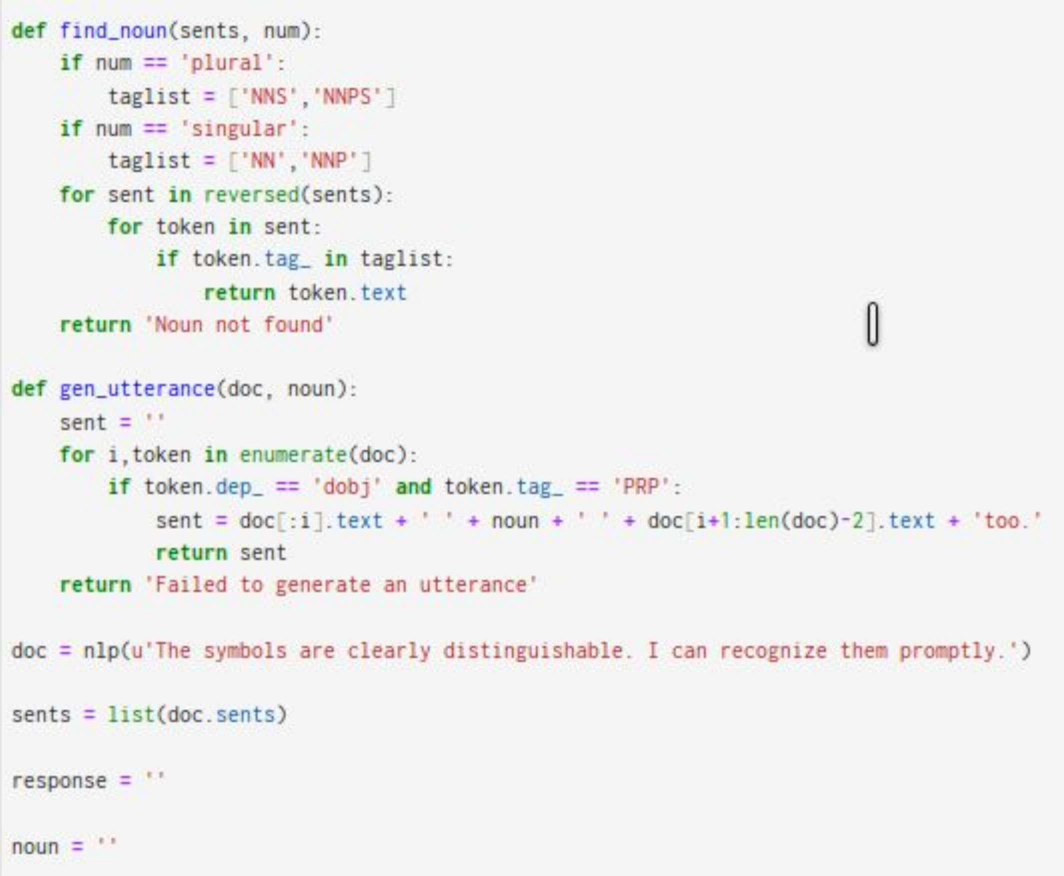
```
def dep_pattern(doc):
    for i in range(len(doc)-1):
        if doc[i].dep_ == 'nsubj' and doc[i+1].dep_ == 'aux' and  doc[i+2].dep_ == 'ROOT':
            for tok in doc[i+2].children:
                if tok.dep_ == 'dobj':
                    return True
    return False

if dep_pattern(doc1):
  print('Found')
else:
  print('Not found')
```

```
Found
```

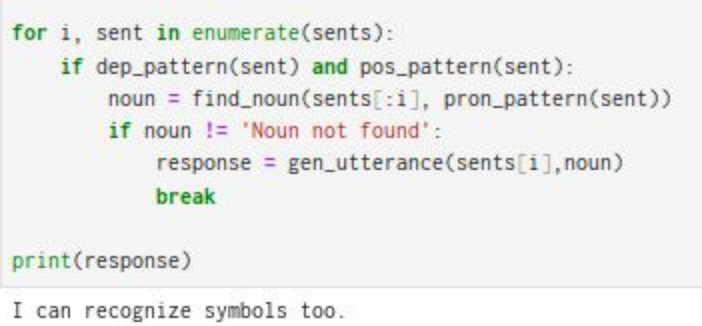
Generating Text



```python
def dep_pattern(doc):
    for i in range(len(doc)-1):
        if doc[i].dep_ == 'nsubj' and doc[i+1].dep_ == 'aux' and  doc[i+2].dep_ == 'ROOT':
            for tok in doc[i+2].children:
                if tok.dep_ == 'dobj':
                    return True
    return False

def pos_pattern(doc):
    for token in doc:
        if token.dep_ == 'nsubj' and token.tag_ != 'PRP':
            return False
        if token.dep_ == 'aux' and token.tag_ != 'MD':
            return False
        if token.dep_ == 'ROOT' and token.tag_ != 'VB':
            return False
        if token.dep_ == 'dobj' and token.tag_ != 'PRP':
            return False
    return True

def pron_pattern(doc):
    plural = ['we','us','they','them']
    for token in doc:
        if token.dep_ == 'dobj' and token.tag_ == 'PRP':
            if token.text in plural:
                return 'plural'
            else:
                return 'singular'
    return 'not found'
```



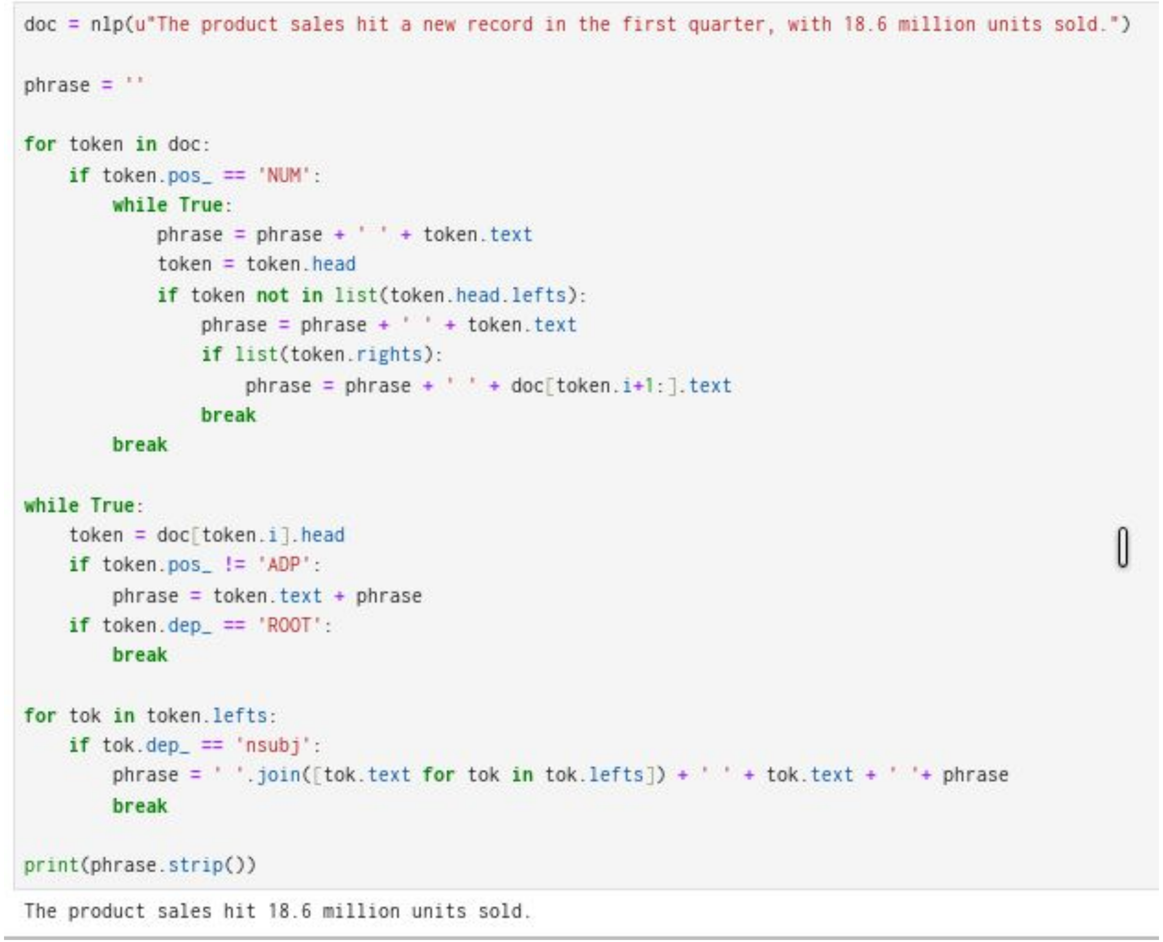
```python
def find_noun(sents, num):
    if num == 'plural':
        taglist = ['NNS','NNPS']
    if num == 'singular':
        taglist = ['NN','NNP']
    for sent in reversed(sents):
        for token in sent:
            if token.tag_ in taglist:
                return token.text
    return 'Noun not found'

def gen_utterance(doc, noun):
    sent = ''
    for i,token in enumerate(doc):
        if token.dep_ == 'dobj' and token.tag_ == 'PRP':
            sent = doc[:i].text + ' ' + noun + ' ' + doc[i+1:len(doc)-2].text + 'too.'
            return sent
    return 'Failed to generate an utterance'

doc = nlp(u'The symbols are clearly distinguishable. I can recognize them promptly.')

sents = list(doc.sents)

response = ''

noun = ''
```



```python
for i, sent in enumerate(sents):
    if dep_pattern(sent) and pos_pattern(sent):
        noun = find_noun(sents[:i], pron_pattern(sent))
        if noun != 'Noun not found':
            response = gen_utterance(sents[i],noun)
            break

print(response)
```

```
I can recognize symbols too.
```

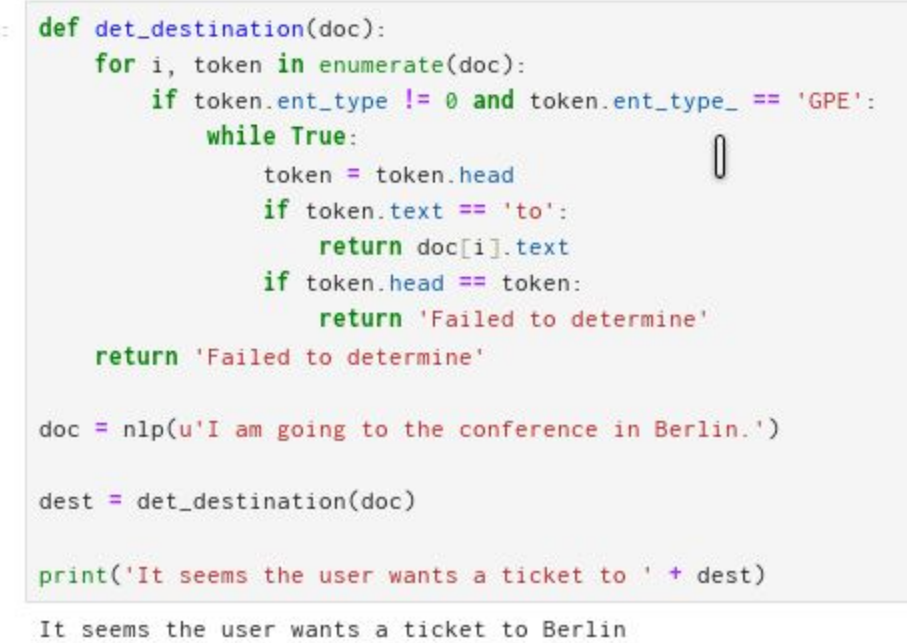
Text Summarization



```python
doc = nlp(u"The product sales hit a new record in the first quarter, with 18.6 million units sold.")

phrase = ''

for token in doc:
    if token.pos_ == 'NUM':
        while True:
            phrase = phrase + ' ' + token.text
            token = token.head
            if token not in list(token.head.lefts):
                phrase = phrase + ' ' + token.text
                if list(token.rights):
                    phrase = phrase + ' ' + doc[token.i+1:].text
                break
        break

while True:
    token = doc[token.i].head
    if token.pos_ != 'ADP':
        phrase = token.text + phrase
    if token.dep_ == 'ROOT':
        break

for tok in token.lefts:
    if tok.dep_ == 'nsubj':
        phrase = ' '.join([tok.text for tok in tok.lefts]) + ' ' + tok.text + ' '+ phrase
        break

print(phrase.strip())
```

```
The product sales hit 18.6 million units sold.
```

Information Extraction



```
def det_destination(doc):
    for i, token in enumerate(doc):
        if token.ent_type != 0 and token.ent_type_ == 'GPE':
            while True:
                token = token.head
                if token.text == 'to':
                    return doc[i].text
                if token.head == token:
                    return 'Failed to determine'
    return 'Failed to determine'

doc = nlp(u'I am going to the conference in Berlin.')

dest = det_destination(doc)

print('It seems the user wants a ticket to ' + dest)
```

```
It seems the user wants a ticket to Berlin
```

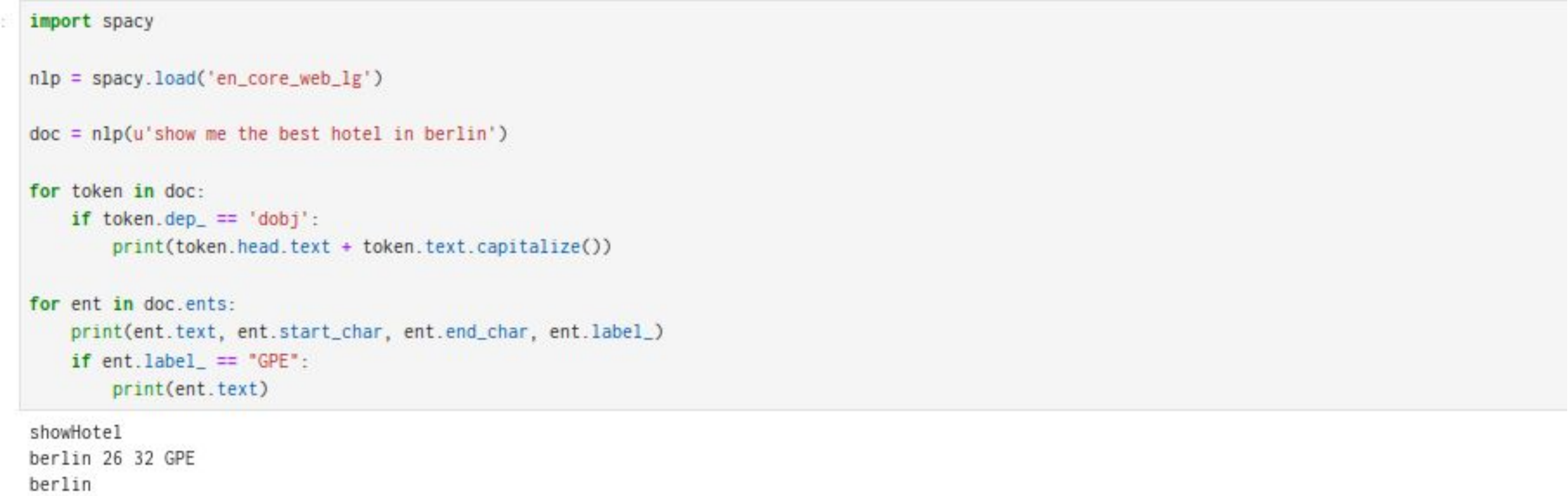
Intention

- Intention is something that is the goal to be achieved.
- Actually, understanding intention is complicated because it is multimodal or revealed from many sides and there are sides that are possibly hidden.
- Recognizing intention in NLP is not intended to reach the level of a human understanding the intention of another human, but only at the level of knowing the intention of what is said / written.

Basically, there are several techniques for recognizing intention:

- Extracting transitive verbs (transitive verbs - verbs that require objects) and direct objects.
- Sentence sequences - Conjunctions
- Recognizing synonyms for different possible intentions
- Using Semantic Similarity
- Recognizing intention from sentence sequences

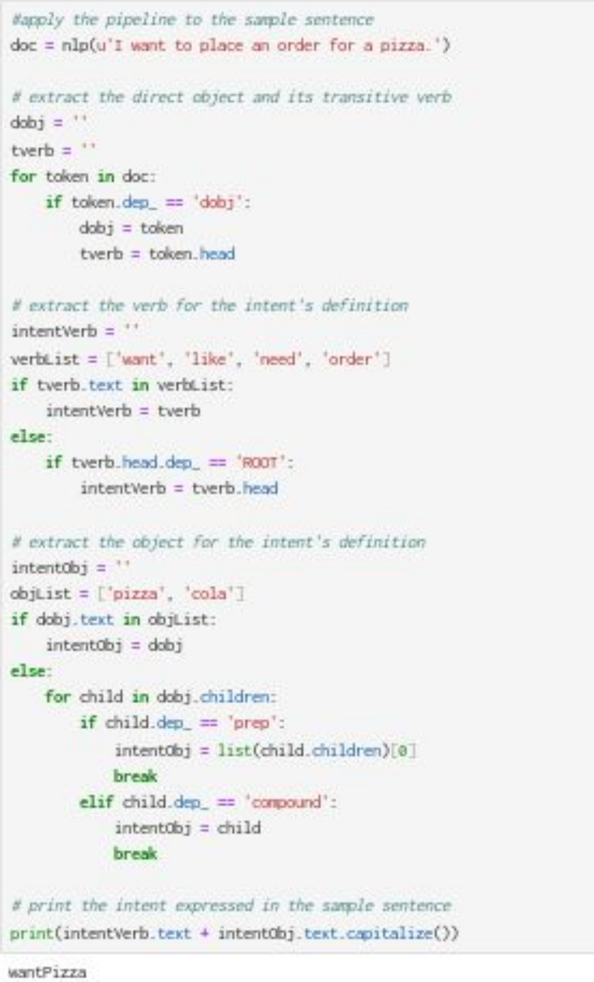
Extracting transitive verbs (transitive verbs - verbs that require objects) and direct objects.



```python
import spacy

nlp = spacy.load('en_core_web_lg')

doc = nlp(u'show me the best hotel in berlin')

for token in doc:
    if token.dep_ == 'dobj':
        print(token.head.text + token.text.capitalize())

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
    if ent.label_ == "GPE":
        print(ent.text)
```

```
showHotel
berlin 26 32 GPE
berlin
```

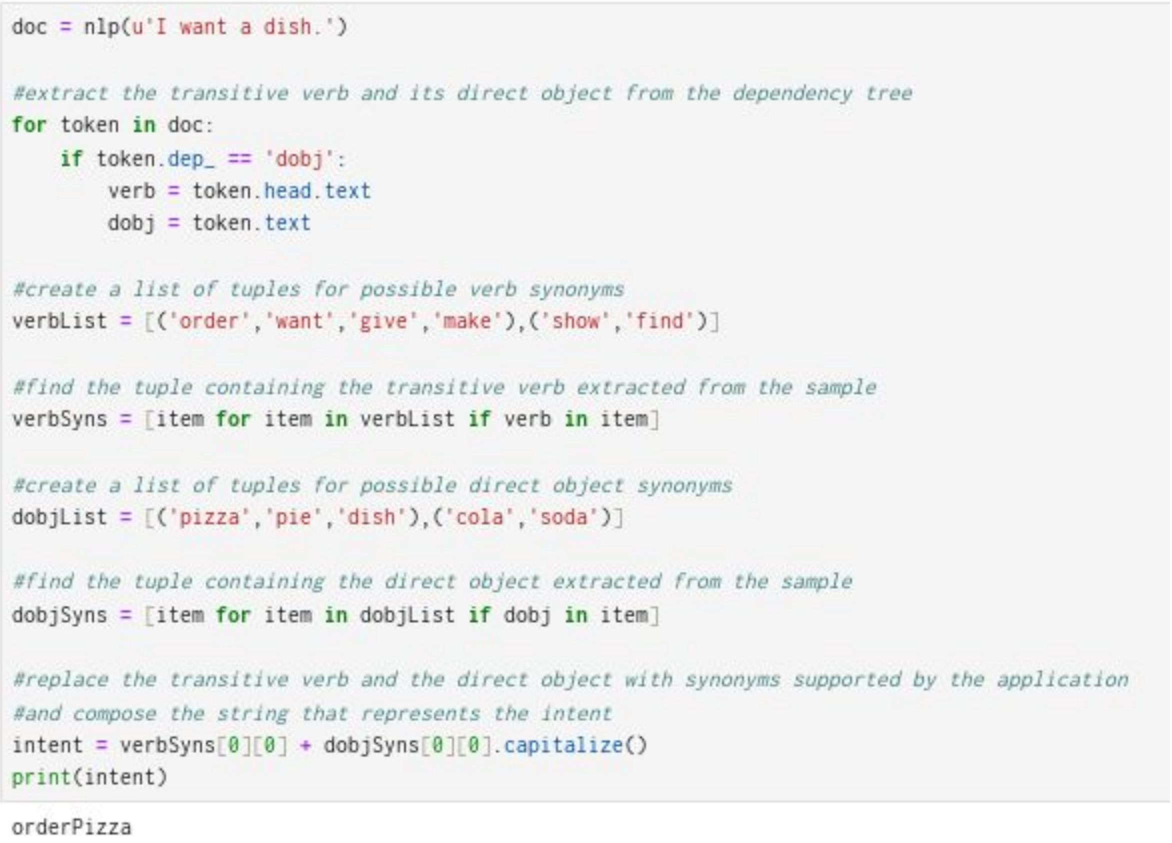
Sentence Sequences - Conjunctions



```python
#apply the pipeline to the sample sentence
doc = nlp(u'I want to place an order for a pizza.')

# extract the direct object and its transitive verb
dobj = ''
tverb = ''
for token in doc:
    if token.dep_ == 'dobj':
        dobj = token
        tverb = token.head

# extract the verb for the intent's definition
intentVerb = ''
verbList = ['want', 'like', 'need', 'order']
if tverb.text in verbList:
    intentVerb = tverb
else:
    if tverb.head.dep_ == 'ROOT':
        intentVerb = tverb.head

# extract the object for the intent's definition
intentObj = ''
objList = ['pizza', 'cola']
if dobj.text in objList:
    intentObj = dobj
else:
    for child in dobj.children:
        if child.dep_ == 'prep':
            intentObj = list(child.children)[0]
            break
        elif child.dep_ == 'compound':
            intentObj = child
            break

# print the intent expressed in the sample sentence
print(intentVerb.text + intentObj.text.capitalize())
```

```
wantPizza
```

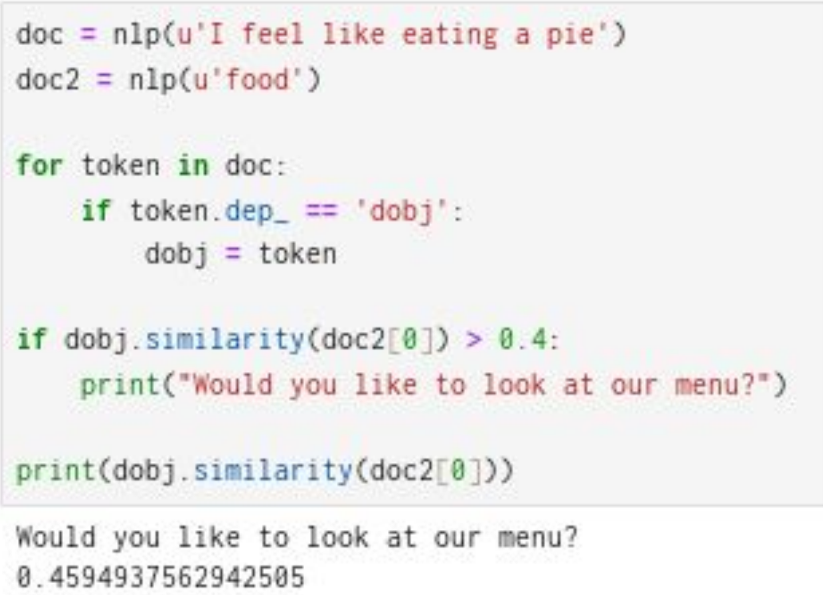
Recognizing Synonyms



```
doc = nlp(u'I want a dish.')

#extract the transitive verb and its direct object from the dependency tree
for token in doc:
    if token.dep_ == 'dobj':
        verb = token.head.text
        dobj = token.text

#create a list of tuples for possible verb synonyms
verbList = [('order','want','give','make'),('show','find')]

#find the tuple containing the transitive verb extracted from the sample
verbSyns = [item for item in verbList if verb in item]

#create a list of tuples for possible direct object synonyms
dobjList = [('pizza','pie','dish'),('cola','soda')]

#find the tuple containing the direct object extracted from the sample
dobjSyns = [item for item in dobjList if dobj in item]

#replace the transitive verb and the direct object with synonyms supported by the application
#and compose the string that represents the intent
intent = verbSyns[0][0] + dobjSyns[0][0].capitalize()
print(intent)
```

```
orderPizza
```

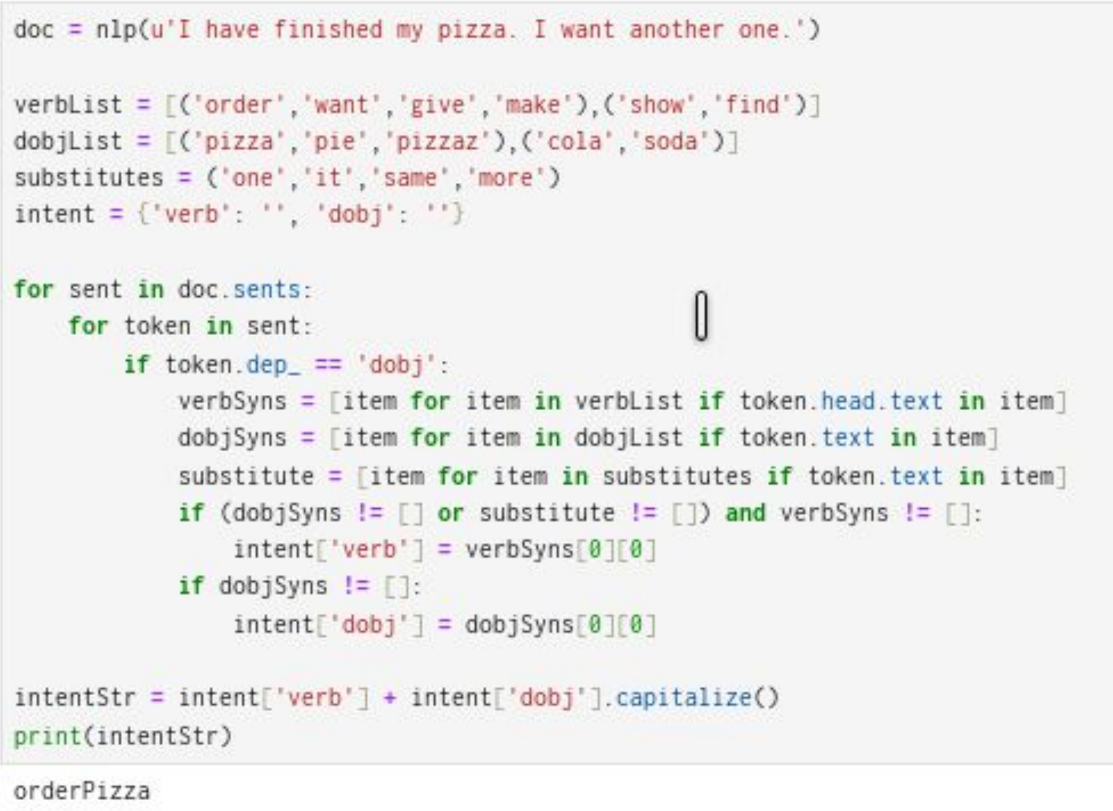
Semantic Similarity



```
doc = nlp(u'I feel like eating a pie')
doc2 = nlp(u'food')

for token in doc:
    if token.dep_ == 'dobj':
        dobj = token

if dobj.similarity(doc2[0]) > 0.4:
    print("Would you like to look at our menu?")

print(dobj.similarity(doc2[0]))
```

```
Would you like to look at our menu?
0.4594937562942505
```

Recognizing intention from sentence sequences



```
doc = nlp(u'I have finished my pizza. I want another one.')

verbList = [('order','want','give','make'),('show','find')]
dobjList = [('pizza','pie','pizzaz'),('cola','soda')]
substitutes = ('one','it','same','more')
intent = {'verb': '', 'dobj': ''}

for sent in doc.sents:
    for token in sent:
        if token.dep_ == 'dobj':
            verbSyns = [item for item in verbList if token.head.text in item]
            dobjSyns = [item for item in dobjList if token.text in item]
            substitute = [item for item in substitutes if token.text in item]
            if (dobjSyns != [] or substitute != []) and verbSyns != []:
                intent['verb'] = verbSyns[0][0]
            if dobjSyns != []:
                intent['dobj'] = dobjSyns[0][0]

intentStr = intent['verb'] + intent['dobj'].capitalize()
print(intentStr)
```

```
orderPizza
```

Large Language Model

- Language Model is a probabilistic model of natural language that can generate probabilities of a series of words based on training in a particular language on a text corpora.
- Probabilistic Model is a statistical approach in the form of quantitative modeling that uses the effects of random events or actions to predict possible future outcomes. Probabilistic models project several possible outcomes that may go beyond what is happening in the current situation.

Language Model categorization:

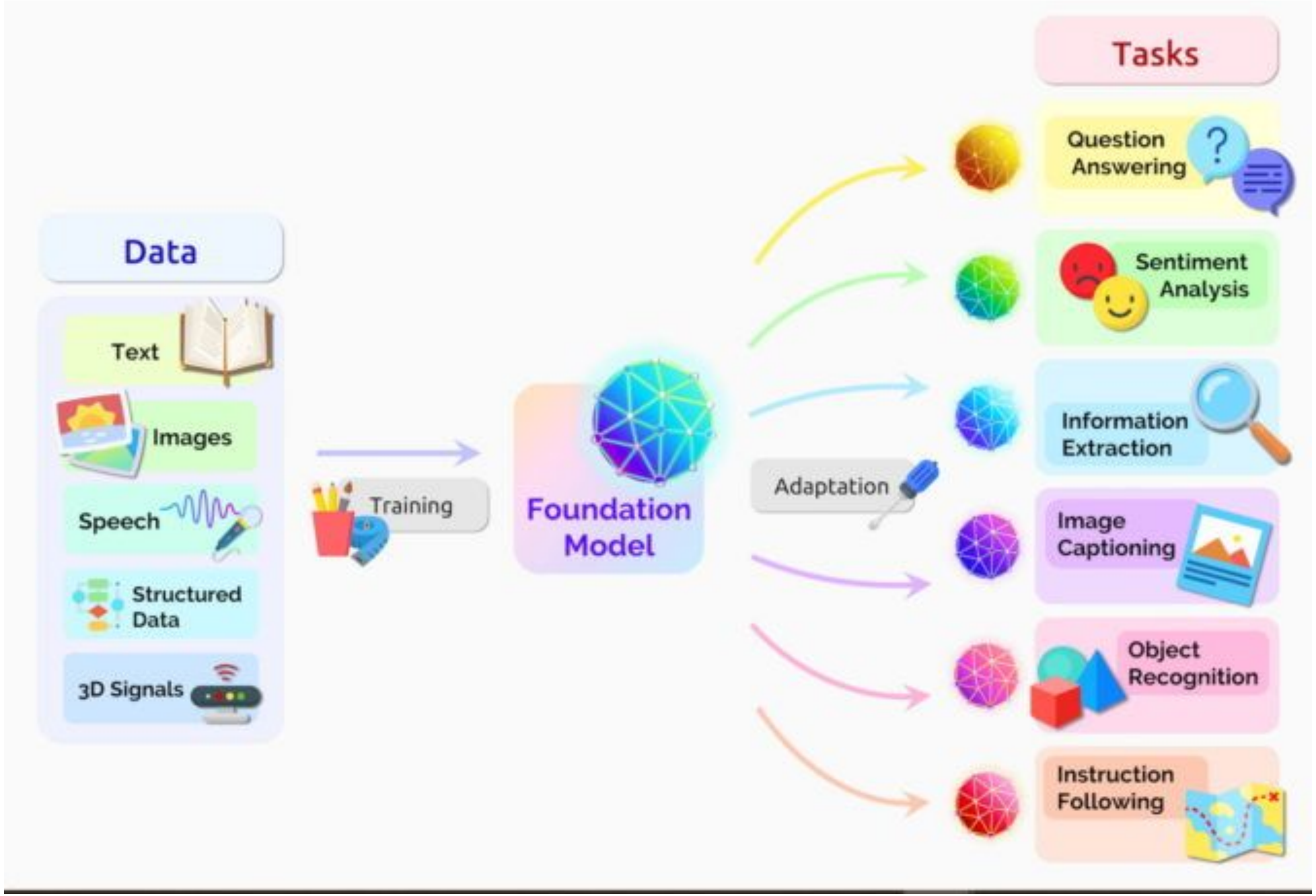
Pure Statistical Model

1. Model berbasis *Word n-grams*
2. Eksponensial
3. Skip-gram

Neural Model

1. *Recurrent Neural Network*: *words embedding*.
2. *Large Language Model* (LLM)

- LLM is a language model that uses ANN (Artificial Neural Network) - specifically using a deep learning architecture called Transformers (more info: https://blogs.nvidia.com/blog/what-is-a-transformer-model/) with large amounts of data to learn billions of parameters during training and uses large computing resources during the training process (both self-supervised and semi-supervised).
- Transformer models are also called *Foundation Model*



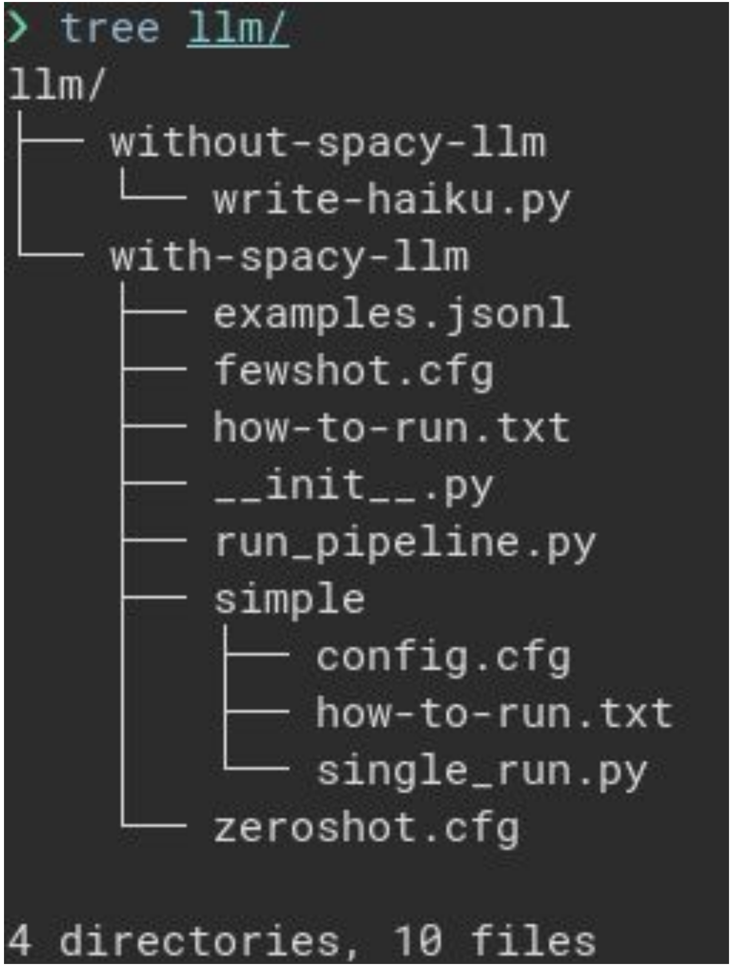
Source: https://blogs.nvidia.com/blog/what-is-a-transformer-model/

LLM Algorithms:

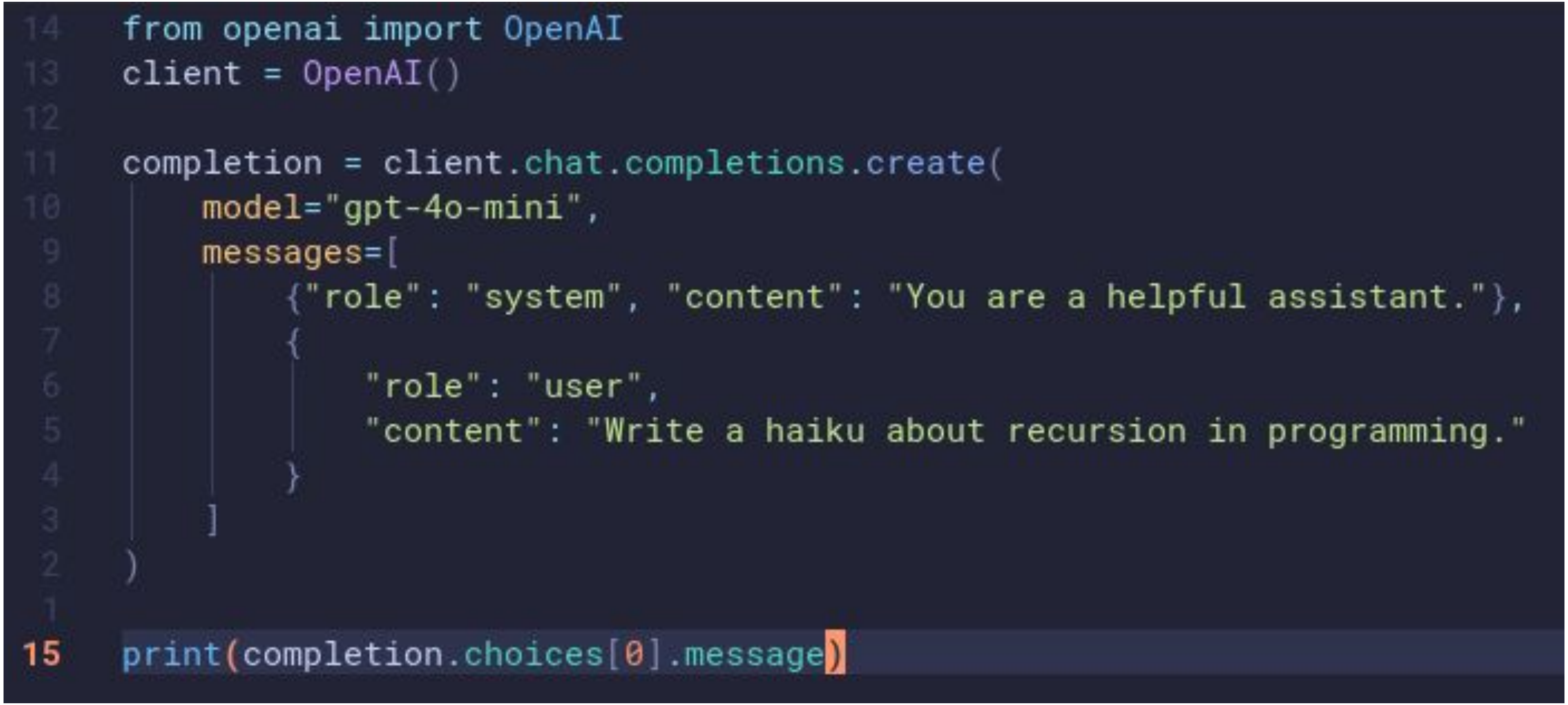
1. PaLM (by Google and is used in Google Bard)
2. GPT (Generative Pre-trained Transformers - OpenAI)
3. LLaMa (from Meta/Facebook)
4. BLOOM (BigScience Large Open-science Open-access Multilingual Language Model)
5. Ernie (Enhanced Representation through Knowledge Integration) from Baidu
6. Anthropic PBC

Note:

- For OpenAI example, you need to have OpenAI API key and organization (OPENAI_API_KEY and OPENAI_API_ORG). You may go to https://openai.com, signup, and get them from your dashboard.
- Also note that you may run out quota if you use free version. You need to pay to OpenAI for more.
- When you already have them, set the environment variables using your shell command before you run your Python script, for example in Bash:
  - export OPENAI_API_KEY=”sk……”
  - export OPENAI_API_ORG=”org….”



```
❯ tree llm/
llm/
├── without-spacy-llm
│   └── write-haiku.py
└── with-spacy-llm
    ├── examples.jsonl
    ├── fewshot.cfg
    ├── how-to-run.txt
    ├── __init__.py
    ├── run_pipeline.py
    ├── simple
    │   ├── config.cfg
    │   ├── how-to-run.txt
    │   └── single_run.py
    └── zeroshot.cfg

4 directories, 10 files
```



```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {
            "role": "user",
            "content": "Write a haiku about recursion in programming."
        }
    ]
)

print(completion.choices[0].message)
```

THANK
YOU!