```
import spacy

#!which python
```

Loading the models and text

trained pipelines:

- download the dutch model (with word vectors): !python -m spacy download nl_core_news_lg
- multi-language: !python -m spacy download xx_sent_ud_sm
- german (accuracy): !python -m spacy download de_dep_news_trf

```
nl_nlp = spacy.load("nl_core_news_lg") #dutch model
de_nlp = spacy.load("de_dep_news_trf") #german model
mul_nlp = spacy.load("xx_sent_ud_sm")  #multilanguage model
```

```
/Users/enriqueviv/Library/Caches/pypoetry/virtualenvs/aesop-spacy-V0v2nuUF-py3.12/lib/python3.12/site-packages/thinc/shims/pytorch.py:253: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load(filelike, map_location=device))
```

```
from spacy.lang.xx import MultiLanguage
# Create a Multilanguage nlp object
#nlp = MultiLanguage()
nlp = spacy.load("nl_core_news_lg")

# Function to select the right model based on the language of the input text
def process_text(text):
    if nl_nlp(text): #checking if it's dutch
        return nl_nlp(text)
    elif de_nlp(text):
        return de_nlp(text) #check if it's german
```

```
    else: #fallback to multilingual model
        return mul_nlp(text)

with open("aesop_extract_nlp_nl_de.txt",
encoding="utf8" ,errors='ignore') as f:
    text = f.read()
```

```
---------------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[145], line 1
----> 1 with open("aesop_extract_nlp_nl_de.txt",
encoding="utf8" ,errors='ignore') as f:
      2     text = f.read()

File ~/Library/Caches/pypoetry/virtualenvs/aesop-spacy-V0v2nuUF-
py3.12/lib/python3.12/site-packages/IPython/core/
interactiveshell.py:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what
you are doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory:
'aesop_extract_nlp_nl_de.txt'
```

```
#check if the text is working by printing first characters
print(text[0:21])
```

```
De Wolf en het Geitje
```

```
# Creating the Doc object
doc = nlp(text)

# Displaying the writing system
print(f"\n{'='*10} Writing System {'='*10}")
print(f"Writing System: {doc.vocab.writing_system}\n")

# Print header for token information
print(f"{'='*10} Token Information {'='*10}\n")
```

```python
# Iterate through the first three tokens and check some attributes
for token in doc[0:3]:
    print(f"Token: {token.text}")
    print(f"  Lemma: {token.lemma_}")
    print(f"  Lex Language: {token.lex.lang_}")
    print(f"  Sentence Span: {token.sent}")
    print(f"  Head: {token.head}")
    print(f"  Index: {token.i}")
    print(f"  POS: {token.pos_}")
    print(f"  Detailed Tag: {token.tag_}")
    print(f"  Dependency: {token.dep_}")
    print(f"  Shape: {token.shape_}")
    print(f"  Is Alpha: {token.is_alpha}")
    print(f"  Is Stop Word: {token.is_stop}")
    print("-" * 30)  # Separator for each token
```

```
========== Writing System ==========
Writing System: {'direction': 'ltr', 'has_case': True, 'has_letters':
True}

========== Token Information ==========

Token: De
  Lemma: De
  Lex Language: nl
  Sentence Span: De Wolf en het Geitje


  Head: De
  Index: 0
  POS: PROPN
  Detailed Tag: SPEC|deeleigen
  Dependency: ROOT
  Shape: Xx
  Is Alpha: True
  Is Stop Word: True
------------------------------
Token: Wolf
  Lemma: Wolf
  Lex Language: nl
  Sentence Span: De Wolf en het Geitje


  Head: De
  Index: 1
  POS: PROPN
  Detailed Tag: SPEC|deeleigen
  Dependency: flat
```

```
  Shape: Xxxx
  Is Alpha: True
  Is Stop Word: False
------------------------------
Token: en
  Lemma: en
  Lex Language: nl
  Sentence Span: De Wolf en het Geitje


  Head: Geitje
  Index: 2
  POS: CCONJ
  Detailed Tag: VG|neven
  Dependency: cc
  Shape: xx
  Is Alpha: True
  Is Stop Word: True
------------------------------
```

```
# A Span object (a slice from the Doc)
span = doc[0:72]
print(span.text)
```

```
De Wolf en het Geitje

Er was eens een klein geitje dat hoorntjes begon te krijgen en daarom dacht dat hij nu al een grote geit was. Hij liep in de wei, samen met zijn moeder en een grote kudde geiten, en zei tegen iedereen dat hij nu wel voor zichzelf kon zorgen. Elke avond gingen de geiten naar hun stal om er te slapen. Op
```

```
# Prepare the data for printing
indices = [token.i for token in doc][6:15]
texts = [token.text for token in doc][6:15]
lemmas = [token.lemma_ for token in doc][6:15]
pos_tags = [token.pos_ for token in doc][6:15]
dependencies = [token.dep_ for token in doc][6:15]
ent_types = [token.ent_type_ if token.ent_type_ else "None" for token
in doc][:15]  # Handle empty entity types
is_alpha = [token.is_alpha for token in doc][6:15]
is_punct = [token.is_punct for token in doc][6:15]
like_num = [token.like_num for token in doc][6:15]
shapes = [token.shape_ for token in doc][6:15]
is_stop = [token.is_stop for token in doc][6:15]

# Print header
print("Lexical Attributes of Tokens:\n" + "=" * 80)
print(f"{'Index':<5} {'Text':<15} {'Lemma':<15} {'POS':<6} {'Dep':<10}
{'Ent Type':<10} {'Shape':<8} {'Is Alpha':<8} {'Is Punct':<8} {'Like
```

```
Num':<10} {'Is Stop'}")
print("-" * 100)

# Print each token's information in a structured format
for index, text, lemma, pos, dep, ent_type, shape, alpha, punct, num,
stop in zip(indices, texts, lemmas, pos_tags, dependencies, ent_types,
shapes, is_alpha, is_punct, like_num, is_stop):
    print(f"{index:<5} {text:<15} {lemma:<15} {pos:<6} {dep:<10}
{ent_type:<10} {shape:<8} {str(alpha):<8} {str(punct):<8}
{str(num):<10} {str(stop)}")

print("=" * 80)
```

```
Lexical Attributes of Tokens:
================================================================================
==========
Index Text            Lemma           POS    Dep        Ent Type
Shape    Is Alpha Is Punct Like Num   Is Stop
--------------------------------------------------------------------------------
------------------------------
6     Er              er              ADV    advmod     None       Xx
True     False    False      True
7     was             zijn            VERB   ROOT       None       xxx
True     False    False      True
8     eens            eens            ADV    advmod     None
xxxx     True     False    False      True
9     een             een             DET    det        None       xxx
True     False    True       True
10    klein           klein           ADJ    amod       None
xxxx     True     False    False      False
11    geitje          gei             NOUN   nsubj      None
xxxx     True     False    False      False
12    dat             dat             DET    det        None       xxx
True     False    False      True
13    hoorntjes       hoornt          NOUN   nsubj      None
xxxx     True     False    False      False
14    begon           beginnen        VERB   csubj      None
xxxx     True     False    False      False
================================================================================
==========
```

```
for ent in doc.ents:
    print(f"{ent.text}, {ent.label_}, {spacy.explain(ent.label_)}")
```

```
Alstublieft, ORG, Companies, agencies, institutions, etc.
vele jaren, DATE, Absolute or relative dates or periods
Hou deze stok stevig vast met je bek en we zullen je zo hoog meenemen
in de lucht dat je het hele land kan zien., WORK_OF_ART, Titles of
books, songs, etc.
twee, CARDINAL, Numerals that do not fall under another type
```

Schildpadden, LOC, Non-GPE locations, mountain ranges, bodies of water
zei de Schildpad, WORK_OF_ART, Titles of books, songs, etc.
Muizen, LOC, Non-GPE locations, mountain ranges, bodies of water
Muizen, GPE, Countries, cities, states
Muizen, LOC, Non-GPE locations, mountain ranges, bodies of water
Kat, WORK_OF_ART, Titles of books, songs, etc.
bel, NORP, Nationalities or religious or political groups
Wie gaat de kat, WORK_OF_ART, Titles of books, songs, etc.
één, CARDINAL, Numerals that do not fall under another type
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
dat., GPE, Countries, cities, states
Stadsmuis, FAC, Buildings, airports, highways, bridges, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
volgende morgen, DATE, Absolute or relative dates or periods
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
volgende morgen, DATE, Absolute or relative dates or periods
Stadsmuis, FAC, Buildings, airports, highways, bridges, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
Kat, WORK_OF_ART, Titles of books, songs, etc.
Doodsbang, ORG, Companies, agencies, institutions, etc.
Veldmuis, FAC, Buildings, airports, highways, bridges, etc.
Stadsmuis, FAC, Buildings, airports, highways, bridges, etc.
nederig, PERSON, People, including fictional
De Vos, PERSON, People, including fictional
eerste, ORDINAL, "first", "second", etc.
Kraanvogel, FAC, Buildings, airports, highways, bridges, etc.
Een Wolf, EVENT, Named hurricanes, battles, wars, sports events, etc.
Kraanvogel, FAC, Buildings, airports, highways, bridges, etc.
de Wolf, PERSON, People, including fictional
Kraanvogel, FAC, Buildings, airports, highways, bridges, etc.
de Wolf, PERSON, People, including fictional
Kraanvogel, FAC, Buildings, airports, highways, bridges, etc.
Wat!, WORK_OF_ART, Titles of books, songs, etc.
snauwde, NORP, Nationalities or religious or political groups
de Wolf, PERSON, People, including fictional
Muis, PERSON, People, including fictional
Leeuw., FAC, Buildings, airports, highways, bridges, etc.
Alstublieft, ORG, Companies, agencies, institutions, etc.
Enkele dagen later, DATE, Absolute or relative dates or periods
één, CARDINAL, Numerals that do not fall under another type
Leeuw, FAC, Buildings, airports, highways, bridges, etc.
Leeuw van dienst, FAC, Buildings, airports, highways, bridges, etc.
Op een avond, WORK_OF_ART, Titles of books, songs, etc.
Trots, PERSON, People, including fictional
Kijk, PERSON, People, including fictional
Wolf, PERSON, People, including fictional

Leeuw op de rug van de Wolf, FAC, Buildings, airports, highways, bridges, etc.
Koning Leeuw, PERSON, People, including fictional
Aap, PERSON, People, including fictional
Koning Leeuw, PERSON, People, including fictional
Op een dag, WORK_OF_ART, Titles of books, songs, etc.
Haas, PERSON, People, including fictional
Geraak, EVENT, Named hurricanes, battles, wars, sports events, etc.
De Haas, WORK_OF_ART, Titles of books, songs, etc.
Schildpad, LOC, Non-GPE locations, mountain ranges, bodies of water
De Vos, PERSON, People, including fictional
Schildpad, LOC, Non-GPE locations, mountain ranges, bodies of water
Haas, PERSON, People, including fictional
Haas, PERSON, People, including fictional
Haas, PERSON, People, including fictional
Eine, GPE, Countries, cities, states
saßen an einem Herbstabend, PERSON, People, including fictional
miteinander, PERSON, People, including fictional
der Spatz, PERSON, People, including fictional
Freundin, PERSON, People, including fictional
Laub, LOC, Non-GPE locations, mountain ranges, bodies of water
heran, ORG, Companies, agencies, institutions, etc.
Blicke, ORG, Companies, agencies, institutions, etc.
Trauben, LOC, Non-GPE locations, mountain ranges, bodies of water
Vorsichtig, GPE, Countries, cities, states
Dann stützte, PERSON, People, including fictional
den Stamm, PERSON, People, including fictional
Trauben erwischen, WORK_OF_ART, Titles of books, songs, etc.
Aber, GPE, Countries, cities, states
Etwas verärgert, ORG, Companies, agencies, institutions, etc.
Mal bemühte, ORG, Companies, agencies, institutions, etc.
sprang aus, WORK_OF_ART, Titles of books, songs, etc.
Leibeskräften, ORG, Companies, agencies, institutions, etc.
Gier, PERSON, People, including fictional
Trauben, PERSON, People, including fictional
den Rücken, ORG, Companies, agencies, institutions, etc.
Blatt, PERSON, People, including fictional
hatte sich bewegt, PERSON, People, including fictional
Der Spatz, PERSON, People, including fictional
schweigend zugesehen hatte, PERSON, People, including fictional
zwitscherte, PERCENT, Percentage, including "%"
Herr Fuchs, PERSON, People, including fictional
hoch hinaus, PERSON, People, including fictional
Die Maus, WORK_OF_ART, Titles of books, songs, etc.
äugte, PERSON, People, including fictional
piepste vorwitzig, WORK_OF_ART, Titles of books, songs, etc.
Mühe, PERSON, People, including fictional
Trauben, LOC, Non-GPE locations, mountain ranges, bodies of water
du nie, PERSON, People, including fictional

Loch zurück, GPE, Countries, cities, states
Fuchs, PERSON, People, including fictional
biß, ORG, Companies, agencies, institutions, etc.
Zähne, PERSON, People, including fictional
Nase, ORG, Companies, agencies, institutions, etc.
meinte, GPE, Countries, cities, states
Trauben, PERSON, People, including fictional
Mit erhobenem Haupt stolzierte er in den Wald zurück, WORK_OF_ART, Titles of books, songs, etc.
Stadt-, PERSON, People, including fictional
Landmaus, NORP, Nationalities or religious or political groups
Eine Landmaus hatte, ORG, Companies, agencies, institutions, etc.
Freundin, PERSON, People, including fictional
eine Stadtmaus, PERSON, People, including fictional
Mangel der sehr verwöhnten Städterin, PERSON, People, including fictional
hatte sie alles, PERSON, People, including fictional
Erbsen, PERSON, People, including fictional
Traubenkerne, GPE, Countries, cities, states
Hafer, PERSON, People, including fictional
wovon, GPE, Countries, cities, states
Landmaus, ORG, Companies, agencies, institutions, etc.
sie, CARDINAL, Numerals that do not fall under another type
Freundin, PERSON, People, including fictional
zuzusprechen., ORG, Companies, agencies, institutions, etc.
Aber, GPE, Countries, cities, states
Stadtmaus, PERSON, People, including fictional
gewohnten Leckereien verwöhnt, PERSON, People, including fictional
Speisen, PERSON, People, including fictional
Gastgeberin, GPE, Countries, cities, states
, sprach sie zu ihr,, WORK_OF_ART, Titles of books, songs, etc.
kümmerlich, GPE, Countries, cities, states
Leben fristest, GPE, Countries, cities, states
Gehe, GPE, Countries, cities, states
Stadt, GPE, Countries, cities, states
Die Landmaus war bald, WORK_OF_ART, Titles of books, songs, etc.
Schnell hatten, PERSON, People, including fictional
Stadt, GPE, Countries, cities, states
führte sie nun in, PERSON, People, including fictional
den Speisesaal, PERSON, People, including fictional
Freundin, PERSON, People, including fictional
Damast, PERSON, People, including fictional
Sessel, GPE, Countries, cities, states
legte, GPE, Countries, cities, states
von den leckeren Speisen, PERSON, People, including fictional
Landmaus nicht, ORG, Companies, agencies, institutions, etc.
Leckerbissen, NORP, Nationalities or religious or political groups
entzückt war, WORK_OF_ART, Titles of books, songs, etc.
Lobsprüche ausbrechen, NORP, Nationalities or religious or political

groups
Freundinnen, NORP, Nationalities or religious or political groups
Landmaus, ORG, Companies, agencies, institutions, etc.
Mühe, PERSON, People, including fictional
eine Ecke, PERSON, People, including fictional
Kaum, PRODUCT, Objects, vehicles, foods, etc. (not services)
Dienerschaft entfernt, LOC, Non-GPE locations, mountain ranges, bodies of water
Freundin, PERSON, People, including fictional
Lebe, PERSON, People, including fictional
Einmal, GPE, Countries, cities, states
Lieber will ich meine, WORK_OF_ART, Titles of books, songs, etc.
Frieden genießen, PERSON, People, including fictional
Speisen, PERSON, People, including fictional
Genügsamkeit und Zufriedenheit, WORK_OF_ART, Titles of books, songs, etc.
Reichtum, LOC, Non-GPE locations, mountain ranges, bodies of water
Ein Wolf, GPE, Countries, cities, states
verschlang, PERSON, People, including fictional
Not setzte, WORK_OF_ART, Titles of books, songs, etc.
demjenigen eine große Belohnung, PERSON, People, including fictional
Beschwerde, NORP, Nationalities or religious or political groups
Kranich, LOC, Non-GPE locations, mountain ranges, bodies of water
Kur, NORP, Nationalities or religious or political groups
höhnte der Wolf, ORG, Companies, agencies, institutions, etc.
du Unverschämter!, GPE, Countries, cities, states
Gehe heim, PRODUCT, Objects, vehicles, foods, etc. (not services)
es meiner Milde, PERSON, People, including fictional
Hilf gern in der Not, WORK_OF_ART, Titles of books, songs, etc.
Dank von einem Bösewichte, PERSON, People, including fictional
Löwe und das Mäuschen

Ein Mäuschen lief über einen schlafenden, WORK_OF_ART, Titles of books, songs, etc.
Löwen, DATE, Absolute or relative dates or periods
Verzeihe, LANGUAGE, Any named language
meine Unvorsichtigkeit, PERSON, People, including fictional
Großmütig, PERSON, People, including fictional
Kurze Zeit, ORG, Companies, agencies, institutions, etc.
Loche, WORK_OF_ART, Titles of books, songs, etc.
Löwen, PERSON, People, including fictional
von wo der Schall, PERSON, People, including fictional
Wohltäter, GPE, Countries, cities, states
einem, GPE, Countries, cities, states
Netze, GPE, Countries, cities, states
Sogleich eilte, PERSON, People, including fictional
Netzes, ORG, Companies, agencies, institutions, etc.
Löwe, PERSON, People, including fictional
Tatzen das übrige zerreißen, WORK_OF_ART, Titles of books, songs, etc.

```
erwiesene Großmut, PERSON, People, including fictional
Selbst unbedeutende Menschen können, WORK_OF_ART, Titles of books,
songs, etc.
Wohltaten, ORDINAL, "first", "second", etc.
Wucher vergelten, ORG, Companies, agencies, institutions, etc.
behandle, ORG, Companies, agencies, institutions, etc.
den Geringsten, ORG, Companies, agencies, institutions, etc.
```

```
print("hash value:", nlp.vocab.strings["terwijl"])
print("string value:", nlp.vocab.strings[6052211810934134087])
```

```
hash value: 6052211810934134087
string value: terwijl
```

```
lexeme = nlp.vocab["terwijl"]
print(f"{lexeme.text}, {lexeme.orth}, {lexeme.is_alpha}")
```

```
terwijl, 6052211810934134087, True
```

A lexeme object is an entry in the vocabulary, contains the context-independent information about a word. Lexemes don't have part-of-speech tags, dependencies or entity labels. Those depend on context

Alphabetical listing

- ADJ: adjective
- ADP: adposition
- ADV: adverb
- AUX: auxiliary
- CCONJ: coordinating conjunction
- DET: determiner
- INTJ: interjection
- NOUN: noun
- NUM: numeral
- PART: particle
- PRON: pronoun
- PROPN: proper noun
- PUNCT: punctuation
- SCONJ: subordinating conjunction
- SYM: symbol
- VERB: verb
- X: other

```
from spacy import displacy
options = {"compact":True, "bg": "#FFA500", "color": "white", "font":
"Helvetica" }
displacy.render(doc, style="dep", options=options)
```

```
<IPython.core.display.HTML object>

#Word vectors
print(doc.vector)

[ 5.47328711e-01 -1.49188650e+00 -4.14488763e-01 -9.44018513e-02
  3.01032156e-01 -2.12837756e-01 -3.64143014e-01 -9.19310868e-01
 -7.59334862e-01 -8.69713485e-01 -5.83031118e-01  9.24258411e-01
 -2.97326416e-01 -1.59773591e-03 -1.55398726e-01  1.46769196e-01
 -8.50905403e-02  1.96294501e-01  3.43021572e-01 -1.15186787e+00
  1.46668404e-01  3.52123141e-01 -8.07253838e-01 -1.52224553e+00
 -6.32632911e-01  9.53147948e-01 -1.86742336e-01 -4.00744319e-01
 -1.04329491e+00  2.13138849e-01  4.29337323e-01  6.35425329e-01
  1.17645931e+00 -1.82646051e-01  1.34384157e-02  7.33261585e-01
 -5.05581200e-01 -4.41793412e-01 -8.96439373e-01  5.18622637e-01
 -1.27144372e+00 -2.59016789e-02  1.41452670e-01 -7.00510442e-01
  7.18112350e-01 -9.42327499e-01  3.92490268e-01 -1.75664164e-02
  5.52664697e-01  5.07134914e-01 -1.71100211e+00  1.20898092e+00
  3.69260043e-01  1.58954784e-01 -2.35649750e-01 -6.12371087e-01
 -2.08470244e-02  4.09601629e-01  9.58297551e-01 -3.85129005e-01
 -1.29386139e+00 -1.36256889e-01 -4.40423250e-01  1.31248713e+00
  1.00969553e+00  1.16692841e+00 -6.06540218e-02  4.73022163e-02
 -5.47070324e-01 -2.88650095e-01 -1.07323611e+00 -1.56560731e+00
 -6.79002464e-01  4.91193414e-01  6.40054464e-01  8.09296489e-01
  1.15825498e+00 -1.17267266e-01  4.95152950e-01 -1.48254007e-01
 -6.94899559e-01 -1.11402023e+00  2.04871148e-01 -1.58971405e+00
  9.36842978e-01  1.73053831e-01 -5.96150160e-01  4.29905415e-01
 -2.50899941e-01  1.82575107e+00  1.09958160e+00 -2.69079834e-01
  9.61110890e-01  1.68756574e-01  9.33075249e-01 -5.18171310e-01
 -3.37741166e-01 -3.45882714e-01  1.62236631e+00 -7.79828787e-01
 -2.95382887e-01  7.80517876e-01  5.23681521e-01 -6.02305055e-01
 -1.70825168e-01 -1.06166400e-01 -1.50924534e-01  1.26873463e-01
  2.86279619e-01  6.41297877e-01  1.07073152e+00  3.25379789e-01
 -1.13759422e+00 -7.14703023e-01 -3.28178048e-01 -8.52149785e-01
 -2.48396784e-01  4.09859508e-01  1.74583042e+00 -1.28506267e+00
  7.09573209e-01  1.00379801e+00  3.28707159e-01  3.66053760e-01
 -1.18598449e+00  7.26538360e-01 -7.92178333e-01  1.58639267e-01
 -6.89976215e-01 -5.91008306e-01  4.45471585e-01  1.16385482e-01
 -5.95624506e-01  1.12044322e+00  2.74877757e-01 -1.84692949e-01
 -4.48052019e-01 -4.35673892e-01  1.44387472e+00  1.99515268e-01
 -1.00341983e-01  5.82119636e-02 -1.52969092e-01 -1.03106344e+00
  1.31626308e+00  2.77389735e-02  1.79216236e-01 -7.03625023e-01
 -4.32271391e-01 -2.78481632e-01  4.77306157e-01 -6.62479818e-01
 -9.88583803e-01  1.11699402e+00 -9.33489725e-02 -6.05678521e-02
  4.23791081e-01  8.24153721e-01  5.05651414e-01 -5.85024953e-01
 -1.23726153e+00 -4.31097984e-01 -7.21452594e-01 -1.46848977e+00
  8.54578018e-01  5.70679426e-01 -6.58399701e-01  9.15151298e-01
  6.32099286e-02  2.45065302e-01  7.25395381e-02  6.56688213e-01
 -1.31147194e+00  1.05285972e-01  9.30610299e-01 -7.29900956e-01
 -3.63356262e-01 -3.06624264e-01  2.98115257e-02  9.49616015e-01
```

```
 -8.55490327e-01  1.81074440e+00 -1.28050268e+00  1.92316711e-01
  7.92748809e-01  7.97147572e-01 -1.31493434e-01  7.93396607e-02
 -1.08638689e-01  7.51428664e-01  1.49787617e+00 -1.32613027e+00
  2.35547945e-01  2.57367134e-01 -2.14144993e+00  6.35172009e-01
  4.94009465e-01  8.25915694e-01  6.64902478e-02 -8.13115120e-01
 -8.82725716e-01 -1.26336277e+00 -5.21018505e-01  7.19773412e-01
 -3.79281342e-01  4.20504838e-01 -6.34424686e-02 -2.49784533e-02
  1.34432018e+00 -1.99714780e-01  5.89839756e-01  3.87085289e-01
  9.44102705e-01 -1.64195403e-01 -9.28564891e-02  4.03227150e-01
  3.45594645e-01  3.57783020e-01 -7.19063699e-01 -2.24941283e-01
 -1.01399040e+00  2.91788995e-01  5.28157294e-01 -1.21664740e-01
  1.13711965e+00 -6.34408474e-01  5.66903651e-01 -6.60739839e-01
  3.14349085e-01 -1.30457842e+00 -5.39543569e-01 -1.10515490e-01
 -1.64475179e+00  7.23927557e-01 -2.77300715e-01 -1.25933945e+00
  1.36155689e+00 -1.72611976e+00 -8.33580911e-01  2.21114218e-01
 -4.08948809e-01 -1.88491213e+00  1.01313579e+00 -9.82259333e-01
 -5.59007406e-01 -7.58207858e-01  4.03517425e-01 -1.21176636e+00
  5.00464916e-01 -1.19461909e-01 -1.20339489e+00 -1.28837967e+00
  9.33053195e-01  1.52306065e-01  5.06786287e-01  6.88277334e-02
  6.34276032e-01 -6.81350112e-01  1.08397312e-01  5.47857881e-01
 -1.99329242e-01 -8.82834718e-02 -5.26448667e-01 -2.30362579e-01
  1.66484371e-01 -9.25745547e-01 -2.58709192e-01 -7.83694088e-01
  7.30440497e-01 -5.02115846e-01 -4.80472803e-01 -1.14923561e+00
 -4.38266955e-02  1.67165190e-01  1.74451008e-01 -6.54905215e-02
 -1.55804265e+00  1.16042562e-01  5.25895953e-01  4.13197607e-01
  4.19330478e-01  2.73330510e-01 -4.13817912e-01  1.29271114e+00
  1.89938307e-01  6.80739224e-01  8.81550387e-02 -6.23259246e-01
  2.94823527e-01 -2.81236500e-01 -1.60225964e+00 -3.52248847e-02
 -9.90399420e-02 -4.82733637e-01 -3.28113735e-01 -1.59558165e+00
 -9.69190896e-01 -7.77614236e-01  2.28195459e-01  9.81518924e-01]
```

```
displacy.render(doc[0:300], style="ent")
```

```
<IPython.core.display.HTML object>
```

```
#Word vectors and semantic similarity (2)
fabel_1 = doc[0:392].text + ". Laat je noot afleiden als je iets wilt bereiken"
print(fabel_1)
```

```
De Wolf en het Geitje

Er was eens een klein geitje dat hoorntjes begon te krijgen en daarom dacht dat hij nu al een grote geit was. Hij liep in de wei, samen met zijn moeder en een grote kudde geiten, en zei tegen iedereen dat hij nu wel voor zichzelf kon zorgen. Elke avond gingen de geiten naar hun stal om er te slapen. Op een avond bleef het klein geitje op de wei staan.  Zijn moeder riep hem om mee naar huis te gaan. Maar hij wilde niet luisteren en bleef knabbelen aan het malse gras. Toen hij na een tijdje rondkeek zag hij dat zijn moeder en de andere geiten al naar
```

```
huis waren.  Hij was helemaal alleen. De zon ging onder en er kropen lange schaduwen over de grond. Een koude wind stak op en maakte akelige geluiden in het gras en in de bomen. Het geitje rilde toen het dacht aan de verschrikkelijke wolf. Hij liep snel over de weide en begon te roepen op zijn moeder. Maar hij was nog niet halfweg toen hij, naast een groepje bomen, de wolf zag staan! Het geitje was bang want het wist dat de wolf hem zou opeten. “Alstublieft, mijnheer de wolf” zei hij bevend “Ik weet dat u mij gaat opeten. Maar speel eerst op uw blokfluit een liedje voor mij, want ik wil dansen en vrolijk zijn, zolang als ik kan.” De wolf vond het een leuk idee om eerst een liedje te spelen vooraleer hij het geitje zou opeten.  Hij zette zijn blokfluit aan zijn mond en speelde een liedje. Het geitje begon vrolijk te dansen en rond te springen. Maar het geluid van de blokfluit werd gehoord door de honden die de geiten beschermen. Ze herkenden het liedje van de wolf en begonnen heel hard naar de wei te lopen. De wolf hield ineens op met zijn liedje en liep snel weg.Terwijl de honden achter hem zaten was hij boos op zichzelf omdat hij zo dom was geweest. In plaats van eerst een liedje te spelen had hij beter het geitje onmiddellijk opgegeten. Laat je. Laat je noot afleiden als je iets wilt bereiken

nlp = spacy.load("nl_core_news_lg")

morale_1 = nlp ("Zorg dat je door je fantasieën de realiteit niet vergeet.")
morale_2 = nlp("Aap geen dingen na indien je er geen verstand van heb")

print(morale_1.similarity(morale_2))

0.7394450121748931
```

Similarity is determined using word vectors

- Multi-dimensional meaning representations of words
- Generated using an algorithm like Word2Vec and lots of text
- Can be added to spaCy's pipelines
- Default: cosine similarity, but can be adjusted
- Doc and Span vectors default to average of token vectors
- Short phrases are better than long documents with many irrelevant words

Similarity depends on the application context Useful for many applications: recommendation systems, flagging duplicates etc. There's no objective definition of "similarity" Depends on the context and what application needs to do

```
satz_1= nlp("Een eenvoudig leven met rust en zekerheid is meer waard dan rijkdom temidden van angst en onzekerheid.")
satz_2 = nlp("Genügsamkeit und Zufriedenheit macht glücklicher als
```

```
Reichtum und Überfluß unter großen Sorgen.")
print(satz_1, "<->", satz_2, satz_1.similarity(satz_2))
```

```
Een eenvoudig leven met rust en zekerheid is meer waard dan rijkdom temidden van angst en onzekerheid. <-> Genügsamkeit und Zufriedenheit macht glücklicher als Reichtum und Überfluß unter großen Sorgen. 0.47801923904604243
```

- There's no objective definition of similarity. Whether "I like burgers" and "I like pasta" is similar depends on your application. Both talk about food preferences, which makes them very similar – but if you're analyzing mentions of food, those sentences are pretty dissimilar, because they talk about very different foods.
- The similarity of Doc and Span objects defaults to the average of the token vectors. This means that the vector for "fast food" is the average of the vectors for "fast" and "food", which isn't necessarily representative of the phrase "fast food".
- Vector averaging means that the vector of multiple tokens is insensitive to the order of the words. Two documents expressing the same meaning with dissimilar wording will return a lower similarity score than two documents that happen to contain the same words while expressing different meanings.

```
nlp.analyze_pipes()
```

```
{'summary': {'tok2vec': {'assigns': ['doc.tensor'],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'morphologizer': {'assigns': ['token.morph', 'token.pos'],
   'requires': [],
   'scores': ['pos_acc', 'morph_acc', 'morph_per_feat'],
   'retokenizes': False},
  'tagger': {'assigns': ['token.tag'],
   'requires': [],
   'scores': ['tag_acc'],
   'retokenizes': False},
  'parser': {'assigns': ['token.dep',
    'token.head',
    'token.is_sent_start',
    'doc.sents'],
   'requires': [],
   'scores': ['dep_uas',
    'dep_las',
    'dep_las_per_type',
    'sents_p',
    'sents_r',
    'sents_f'],
   'retokenizes': False},
  'lemmatizer': {'assigns': ['token.lemma'],
   'requires': [],
   'scores': ['lemma_acc'],
```

```
   'retokenizes': False},
  'attribute_ruler': {'assigns': [],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'ner': {'assigns': ['doc.ents', 'token.ent_iob', 'token.ent_type'],
   'requires': [],
   'scores': ['ents_f', 'ents_p', 'ents_r', 'ents_per_type'],
   'retokenizes': False}},
 'problems': {'tok2vec': [],
  'morphologizer': [],
  'tagger': [],
  'parser': [],
  'lemmatizer': [],
  'attribute_ruler': [],
  'ner': []},
 'attrs': {'token.morph': {'assigns': ['morphologizer'], 'requires':
[]},
  'token.tag': {'assigns': ['tagger'], 'requires': []},
  'token.head': {'assigns': ['parser'], 'requires': []},
  'doc.tensor': {'assigns': ['tok2vec'], 'requires': []},
  'token.pos': {'assigns': ['morphologizer'], 'requires': []},
  'token.ent_iob': {'assigns': ['ner'], 'requires': []},
  'token.ent_type': {'assigns': ['ner'], 'requires': []},
  'doc.sents': {'assigns': ['parser'], 'requires': []},
  'doc.ents': {'assigns': ['ner'], 'requires': []},
  'token.lemma': {'assigns': ['lemmatizer'], 'requires': []},
  'token.dep': {'assigns': ['parser'], 'requires': []},
  'token.is_sent_start': {'assigns': ['parser'], 'requires': []}}}
```

```
# compare two documents
fabel1 = nlp()
fabel4= nlp("Een Stadsmuis ging op bezoek bij een familielid, welke in
het veld woonde, en bleef daar eten.De Veldmuis serveerde een maaltijd
van tarwe, wortels en eikels, samen met wat koud water om erbij te
drinken. De Stadsmuis at maar heel weinig en nam slechts een hapje van
dit en een hapje van dat. Het was heel duidelijk dat ze het eenvoudige
eten niet lustte en er alleen maar aan knabbelde om niet onbeleefd te
zijn. Na de maaltijd begon de Stadsmuis te spreken over haar luxe
leven in de stad, terwijl de Veldmuis aandachtig luisterde. Daarna
gingen ze naar bed in een gezellig nestje onder de grond en sliepen
rustig en ongestoord tot de volgende morgen. Terwijl ze sliep droomde
de Veldmuis dat ze een Stadsmuis was en dat ze genoot van alle luxe en
genoegens waarover de Stadsmuis verteld had. De volgende morgen vroeg
de Stadsmuis aan de Veldmuis of ze graag mee ging naar de stad. De
Veldmuis was blij en zei ja. Toen ze in de stad waren gingen ze binnen
in een mooi, groot huis. In de eetkamer stond een tafel met daarop de
overschotjes van een rijkelijk feestmaal. Er waren snoepjes en
gelatinepudding, taartjes, heerlijke kazen, en nog veel andere zaken
die muizen zo graag eten. Maar toen de Veldmuis aan een taartje wou
```

```
knabbelen hoorden ze een Kat luid miauwen en krabben aan de deur.
Doodsbang vluchten de muizen naar een schuilplaats en daar bleven ze
lange tijd heel stil liggen; ze durfden zelfs amper ademhalen.Toen ze
zich tenslotte terug naar de tafel waagden zwaaide plots de deur open.
Er kwamen dienstboden binnen om de tafel af te ruimen, op de voet
gevolgd door de hond van het huis. In paniek vluchtten de muisjes
terug naar hun schuilplaats, welke ze veilig bereikten. Van zodra de
dienstboden en de hond de kamer hadden verlaten nam de Veldmuis haar
paraplu en haar handtas en zei tegen de Stadsmuis: “Je hebt meer luxe
en lekkernijen dan ik heb, maar toch heb ik liever mijn eenvoudig eten
en mijn nederig leventje op het platteland. En vooral de vrede en de
veiligheid die erbij horen.” Een eenvoudig leven met rust en zekerheid
is meer waard dan rijkdom temidden van angst en onzekerheid. ")
print(fabel1.similarity(fabel4))
```

```
0.9625629959186227
```

creating two doc objects and using the first fabel similarity method comparing it with the fourth we get a similarity of 95%

```
#compare two tokens
token_fabel1 = fabel1[2]
#een klein geitje
token_fabel4 = fabel4[6]
#een stadsmuis
print(token_fabel1.similarity(token_fabel4))
print(token_fabel1, token_fabel4)
```

```
0.24153238534927368
eens een
```

```
# compare a document with a token
print(fabel1.similarity(token_fabel1))
```

```
0.5169401537350471
```

The first fable and the token "geitije" is showing a similarity of 0.26

```
#Compare a span with a document
span_fabel1 = fabel1 [4:6]
span_fabel4 = fabel4 [1:22]
#klein geitje , Stadsmuis ging op bezoek bij een familielid, welke in
het veld woonde, en bleef daar eten.De Veldmuis
print(span_fabel1.similarity(fabel1))
print(span_fabel4.similarity(fabel4))
```

```
0.4180270812088566
0.907988325697748
```

The span "klein geitje" would have a simliarity of 0.41 with the first fable, and the span " Stadsmuis ging op bezoek bij een familielid, welke in het veld woonde, en bleef daar eten.De Veldmuis" would have a similarity of 0.90 with the 4th fable.

```
moral_fabel1 = nlp("Laat je nooit afleiden als je iets wilt bereiken")
moral_fabel4 = nlp("Een eenvoudig leven met rust en zekerheid is meer
waard dan rijkdom temidden van angst en onzekerheid")
similarity_moral1_moral4= moral_fabel1.similarity(moral_fabel4)
print(similarity_moral1_moral4)
```

```
0.4098061769465898
```

Creating a span for the morals of the first and fourth fable "Laat je nooit afleiden als je iets wilt bereiken/ Een eenvoudig leven met rust en zekerheid is meer waard dan rijkdom temidden van angst en onzekerheid" we get a similarity of 0.409806

Combining predicitions and rules

How does spaCy predict similarity?

- Similarity is determined using word vectors
- Multi-dimensional meaning representations of words
- Generated using an algorithm like Word2Vec and lots of text
- Can be added to spaCy's pipelines
- Default: cosine similarity, but can be adjusted
- Doc and Span vectors default to average of token vectors
- Short phrases are better than long documents with many irrelevant words

Similarity depends on the application context Useful for many applications: recommendation systems, flagging duplicates etc. There's no objective definition of "similarity" Depends on the context and what application needs to do doc1 = nlp("I like cats") doc2 = nlp("I hate cats")

```
print(doc1.similarity(doc2))
```

- 0.9501447503553421

However, it's important to keep in mind that there's no objective definition of what's similar and what isn't. It always depends on the context and what your application needs to do.

Here's an example: spaCy's default word vectors assign a very high similarity score to "I like cats" and "I hate cats". This makes sense, because both texts express sentiment about cats. But in a different application context, you might want to consider the phrases as very dissimilar, because they talk about opposite sentiments.

```
#word vectors of the first fable in Dutch
print(fabel1.vector)
```

```
[ 1.1044049  -1.5757446  -0.4716865  -0.26124424  0.24331798 -
0.4406284
 -0.470388   -1.1420385  -0.59314394 -1.1364907  -1.0438812
```

```
1.4370047
 -0.0729334  -0.153751    0.01478354  0.19217241 -0.20888656
0.48006728
  0.6001522  -1.4752192  -0.00276125  0.6537229  -0.68655807 -
1.8204771
 -0.4575848   1.0441085  -0.49087888 -0.6916356  -1.2901576
0.01269883
  0.87853235  0.6711759   1.5894083  -0.54994535 -0.00826322
1.2722178
 -0.4322306  -0.7848753  -1.0715708   0.56717896 -1.375841   -
0.08917621
 -0.25009942 -0.63209826  0.8124984  -1.5087636   0.47693658 -
0.19462328
  0.87132865  0.26019284 -1.6007915   1.2812878   0.36760432
0.05930824
 -0.5500791  -1.0771222  -0.19791196  0.87483585  1.3575584  -
0.52950233
 -1.4416133  -0.38478535 -0.668372    1.908296    1.2278684
1.0506487
 -0.11760367  0.1908147  -0.62261903 -0.15109849 -1.2780397  -
2.1158414
 -0.7746295   0.7297804   0.9428004   1.3894616   1.8607532  -
0.17004217
  0.9850815  -0.50974596 -0.87218314 -1.9018618   0.05400246 -
2.1320207
  1.2531372   0.04235886 -0.7100132   0.3571686   0.20919898
2.4566083
  1.258467   -0.24632418  1.6249319   0.2501839   1.2160087  -0.646337
 -0.46339148 -0.1818665   2.1384857  -1.1818243  -0.39926392
0.9373507
  0.80914855 -1.0251778  -0.02300278 -0.336771    0.00713144
0.61802506
  0.16651963  0.4422551   1.3508759   0.28063297 -1.3470054  -
0.76314545
 -0.31242427 -0.75761896 -0.41329825  0.503247    2.267708   -
1.5008856
  0.9578355   1.3214972   0.4640311  -0.20348114 -1.4810811
0.8595321
 -1.1345793   0.01780795 -0.8151055  -0.6223169   0.95698917
0.0101658
 -0.6688305   1.4323175   0.22762991 -0.21241242 -0.62945956 -
0.94752824
  2.3210568   0.13869113  0.00811667  0.3592607  -0.24337782 -1.155734
  1.6728936   0.3220793   0.32919943 -1.234979   -0.49424133 -
0.5338253
  0.321616   -0.90628093 -1.5270606   1.5023905  -0.14988166
0.37297118
  0.20842224  1.332149    0.7111054  -0.3429054  -1.4576999  -
0.47837535
```

```
 -0.9029259  -1.74068     1.1171595   0.47078702 -0.53350914
0.92582506
 -0.01328146  0.36184934  0.119984    0.65203995 -1.5708396
0.15123415
  1.3582861  -0.86175376 -0.4553798  -0.67111033  0.21101186
1.4363074
 -1.1627672   2.7538264  -1.6964599   0.03279154  0.8403761
0.9490069
  0.10119231  0.48658198 -0.11494093  1.1227704   1.511112   -
1.5114187
 -0.17436212  0.14537962 -2.2272012   0.87745136 -0.06106514
0.6824299
  0.19650705 -0.93905926 -0.8718952  -2.0655198  -0.44764885
1.0919604
 -0.6796275   0.6619722  -0.30871528  0.16439007  1.7210062  -
0.36832958
  0.7152932   0.3224934   0.9168273  -0.38441828 -0.1178847
0.4528879
  0.06403878  0.02847821 -0.60037357 -0.17858483 -1.3808645
0.6238168
  0.9750767   0.17647628  1.65933    -0.69169605  0.6261708  -
0.9764877
  0.42180845 -1.4932109  -0.40729687 -0.3390077  -2.1107488
0.45767328
 -0.26387745 -1.6374485   1.5904299  -1.9112915  -0.5522292
0.24381034
 -0.57338136 -2.361247    1.2615491  -1.4853519  -0.66444695 -
1.0678492
  0.25659454 -1.571925    0.6982469  -0.40764415 -1.448042   -
1.4257123
  1.3396169   0.12618434  0.57288826  0.09590071  0.4724039  -
0.4589781
  0.04417466  0.87677    -0.37188503  0.3024965  -0.88770485 -
0.14262757
 -0.06673154 -1.2956102  -0.30534956 -1.2325859   0.9409272  -
0.56129026
 -0.46488085 -1.5509083  -0.19456595  0.21691969  0.29494327 -
0.27482045
 -2.153154    0.05185352  1.3535515   0.42372686  0.73769546
0.48118886
 -0.60083395  1.4742268   0.49895024  0.85982746 -0.0973737  -
1.1672618
  0.18007268 -0.6482361  -2.019692    0.08518007  0.01937998 -
0.9114054
 -0.29588932 -2.1065352  -0.81739056 -0.9648571   0.17040578
1.2633262 ]

from spacy.matcher import Matcher
```

```
#test adding the match  klein geitje as geit
matcher = Matcher(nlp.vocab)
matcher.add("GEIT", [[{"LOWER":"klein"}, {"LOWER":"geitje"}]])
for match_id, start, end in matcher(fabel1):
    span = doc[start:end]
    print("Matched span:", span.text)
    #Get the span's root token and root head token
    print("Root token:", span.root.text)
    print("Root head token:", span.root.head.text)
    # Get the previous token and its POS tag
    print("previous token:", doc[start -1].text, doc[start -1].pos_)
```

```
Matched span: Geitje


Root token: Geitje
Root head token: Geitje
previous token: het
Matched span: . Op
Root token: .
Root head token: .
previous token: slapen
```

```
#efficient phrase matching for Veldmuis and Stadtmuis
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
pattern = nlp("klein geitje")
matcher.add("GEIT",[pattern])
# Iterate over the matches
for match_id, start, end in matcher(doc):
    #get the matched span
    span = fabel1[start:end]
    print("Matched span:", span.text)
```

```
Matched span: krijgen en
Matched span: .
```

```
# grabbing proper nouns
matcher = Matcher(nlp.vocab)
pattern = [{"POS": "PROPN"}]
matcher.add("PROPER_NOUNS", [pattern])
doc = nlp(text)
matches = matcher(doc)
print (len(matches))
for match in matches[:10]:
    print (match, doc[match[1]:match[2]])
```

```
391
(3232560085755078826, 0, 1) De
(3232560085755078826, 1, 2) Wolf
(3232560085755078826, 227, 228) wolf
```

```
(3232560085755078826, 401, 402) De
(3232560085755078826, 402, 403) Schildpad
(3232560085755078826, 534, 535) Haas
(3232560085755078826, 537, 538) Eekhoorn
(3232560085755078826, 653, 654) vallen.”De
(3232560085755078826, 654, 655) Schildpad
(3232560085755078826, 724, 725) de
```

```
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
pattern = nlp("de wolf")
matcher.add("WOLF", [pattern])
# Iterate over the matches
for match_id, start, end in matcher(doc):
    # Get the matched span
    span = doc[start:end]
    print("Matched span:", span.text)
```

```
Matched span: de wolf
Matched span: de wolf
Matched span: de wolf
Matched span: de wolf
Matched span: de wolf
```

```
matcher = Matcher(nlp.vocab)
matcher.add("WOLF", [[{"LOWER": "wolf"}, {"LOWER": "de wolf"}]])
test_doc = nlp(fabel_1)
for match_id, start, end in matcher(test_doc):
    span = test_doc[start:end]
    print("Matched span:", span.text)
    # Get the span's root token and root head token
    print("Root token:", span.root.text)
    print("Root head token:", span.root.head.text)
    # Get the previous token and its POS tag
    print("Previous token:", test_doc[start - 1].text, test_doc[start
- 1].pos_)
```

```
#Improving it with Multi-Word Tokens
matcher = Matcher(nlp.vocab)
pattern = [{"POS": "PROPN", "OP": "+"}]
matcher.add("PROPER_NOUNS", [pattern])
doc = nlp(text)
matches = matcher(doc)
print (len(matches))
for match in matches[:10]:
    print (match, doc[match[1]:match[2]])
```

```
586
(3232560085755078826, 0, 1) De
(3232560085755078826, 0, 2) De Wolf
```

```
(3232560085755078826, 1, 2) Wolf
(3232560085755078826, 227, 228) wolf
(3232560085755078826, 401, 402) De
(3232560085755078826, 401, 403) De Schildpad
(3232560085755078826, 402, 403) Schildpad
(3232560085755078826, 534, 535) Haas
(3232560085755078826, 537, 538) Eekhoorn
(3232560085755078826, 653, 654) vallen.”De
```

```
#Greedy Keyword Argument
matcher = Matcher(nlp.vocab)
pattern = [{"POS": "PROPN", "OP": "+"}]
matcher.add("PROPER_NOUNS", [pattern], greedy='LONGEST')
doc = nlp(text)
matches = matcher(doc)
print (len(matches))
for match in matches[:10]:
    print (match, doc[match[1]:match[2]])
```

```
253
(3232560085755078826, 2970, 2975) Ihr wollt zu hoch hinaus
(3232560085755078826, 3226, 3231) du es in der Stadt
(3232560085755078826, 2835, 2839) sehnsüchtig an den dicken
(3232560085755078826, 3382, 3386) Reste des Mahles zu
(3232560085755078826, 3589, 3593) verdanke es meiner Milde
(3232560085755078826, 3726, 3730) von wo der Schall
(3232560085755078826, 3759, 3763) das übrige zerreißen konnte
(3232560085755078826, 3784, 3788) behandle auch den Geringsten
(3232560085755078826, 2823, 2826) Der Fuchs schlich
(3232560085755078826, 2828, 2831) den Weinstock heran
```

```
# Sorting it to Appearance
matcher = Matcher(nlp.vocab)
pattern = [{"POS": "PROPN", "OP": "+"}]
matcher.add("PROPER_NOUNS", [pattern], greedy='LONGEST')
doc = nlp(text)
matches = matcher(doc)
matches.sort(key = lambda x: x[1])
print (len(matches))
for match in matches[:10]:
    print (match, doc[match[1]:match[2]])
```

```
253
(3232560085755078826, 0, 2) De Wolf
(3232560085755078826, 227, 228) wolf
(3232560085755078826, 401, 403) De Schildpad
(3232560085755078826, 534, 535) Haas
(3232560085755078826, 537, 538) Eekhoorn
(3232560085755078826, 653, 655) vallen.”De Schildpad
(3232560085755078826, 724, 726) de Schildpadden
```

```
(3232560085755078826, 735, 736) Schildpad
(3232560085755078826, 775, 776) Dwaze
(3232560085755078826, 786, 787) Kat

# Adding in Sequences
matcher = Matcher(nlp.vocab)
pattern = [{"POS": "PROPN", "OP": "+"}, {"POS": "VERB"}]
matcher.add("PROPER_NOUNS", [pattern], greedy='LONGEST')
doc = nlp(text)
matches = matcher(doc)
matches.sort(key = lambda x: x[1])
print (len(matches))
for match in matches:
    print (match, doc[match[1]:match[2]])

44
(3232560085755078826, 838, 840) Kat verlost
(3232560085755078826, 1050, 1052) Stadsmuis ging
(3232560085755078826, 1070, 1072) Veldmuis serveerde
(3232560085755078826, 1093, 1095) Stadsmuis at
(3232560085755078826, 1199, 1201) Stadsmuis verteld
(3232560085755078826, 1781, 1783) Kraanvogel had
(3232560085755078826, 1808, 1810) Wolf vroeg
(3232560085755078826, 1814, 1816) Wolf voelde
(3232560085755078826, 1840, 1842) Kraanvogel verontwaardigd
(3232560085755078826, 1916, 1918) Muis kwam
(3232560085755078826, 2016, 2018) Muis gezegd
(3232560085755078826, 2035, 2037) Muis gaan
(3232560085755078826, 2082, 2084) Muis herkende
(3232560085755078826, 2148, 2150) Muis zei
(3232560085755078826, 2441, 2443) Aap stond
(3232560085755078826, 2516, 2518) Leeuw misten
(3232560085755078826, 2614, 2617) De Haas vond
(3232560085755078826, 2716, 2718) Haas lag
(3232560085755078826, 2723, 2725) Haas sliep
(3232560085755078826, 2742, 2745) De Haas begon
(3232560085755078826, 2833, 2835) Blicke hingen
(3232560085755078826, 2845, 2847) Vorsichtig spähte
(3232560085755078826, 2911, 2913) Mal bemühte
(3232560085755078826, 2921, 2923) Gier haschte
(3232560085755078826, 2961, 2965) beherrschen und zwitscherte
belustigt
(3232560085755078826, 3015, 3017) Zähne zusammen
(3232560085755078826, 3043, 3045) Haupt stolzierte
(3232560085755078826, 3077, 3080) Wohnung aufs freundlichste
(3232560085755078826, 3090, 3092) zu lassen
(3232560085755078826, 3186, 3188) Gastgeberin merken
(3232560085755078826, 3188, 3190) zu lassen
(3232560085755078826, 3197, 3199) Geschmack gewesen
(3232560085755078826, 3263, 3266) zum Mitgehen bereit
```

```
(3232560085755078826, 3267, 3269) Schnell hatten
(3232560085755078826, 3271, 3273) Stadt erreicht
(3232560085755078826, 3309, 3311) Stadtmaus führte
(3232560085755078826, 3382, 3387) Reste des Mahles zu verzehren
(3232560085755078826, 3431, 3434) vor Schrecken zitternd
(3232560085755078826, 3434, 3438) zu ihrer Freundin sprach
(3232560085755078826, 3463, 3465) Speisen schwelgen
(3232560085755078826, 3494, 3497) Ein Wolf hatte
(3232560085755078826, 3639, 3642) Der Löwe erwachte
(3232560085755078826, 3764, 3766) So vergalt
(3232560085755078826, 3773, 3775) Selbst unbedeutende
```

```
displacy.render(fabel1, style="ent")
```

```
<IPython.core.display.HTML object>
```

```
displacy.render(fabel4, style="ent")
```

```
<IPython.core.display.HTML object>
```

```
nlp = spacy.blank('nl')
doc = nlp(text)
```

```
from spacy import displacy
displacy.render(doc, style= "dep")
```

```
<IPython.core.display.HTML object>
```

```
displacy.render(doc, style="ent")
```

```
<IPython.core.display.HTML object>
```

```
test_doc1 = nlp("Hilf gern in der Not, erwarte aber keinen Dank von
einem Bösewichte, sondern sei zufrieden, wenn er dich nicht
beschädigt.")
test_doc2 = nlp("Selbst unbedeutende Menschen können bisweilen
Wohltaten mit Wucher vergelten, darum behandle auch den Geringsten
nicht übermütig.")
```

```
import spacy

nlp = spacy.load('de_dep_news_trf')
```

```
test_doc1 = nlp("Hilf gern in der Not, erwarte aber keinen Dank von
einem Bösewichte, sondern sei zufrieden, wenn er dich nicht
beschädigt.")
test_doc2 = nlp("Selbst unbedeutende Menschen können bisweilen
Wohltaten mit Wucher vergelten, darum behandle auch den Geringsten
nicht übermütig.")

similarity = test_doc1.similarity(test_doc2)
print(test_doc1, "<->", test_doc2, similarity)
```

```
Hilf gern in der Not, erwarte aber keinen Dank von einem Bösewichte, sondern sei zufrieden, wenn er dich nicht beschädigt. <-> Selbst unbedeutende Menschen können bisweilen Wohltaten mit Wucher vergelten, darum behandle auch den Geringsten nicht übermütig. 0.0

/Users/enriqueviv/Library/Caches/pypoetry/virtualenvs/aesop-spacy-V0v2nuUF-py3.12/lib/python3.12/site-packages/thinc/shims/pytorch.py:114: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(self._mixed_precision):
/var/folders/8r/1rwtjmt1489bxf0kzr5g5s300000gn/T/ipykernel_9676/941098153.py:8: UserWarning: [W007] The model you're using has no word vectors loaded, so the result of the Doc.similarity method will be based on the tagger, parser and NER, which may not give useful similarity judgements. This may happen if you're using one of the small models, e.g. `en_core_web_sm`, which don't ship with word vectors and only use context-sensitive tensors. You can always add your own word vectors, or use one of the larger models instead if available.
  similarity = test_doc1.similarity(test_doc2)
/var/folders/8r/1rwtjmt1489bxf0kzr5g5s300000gn/T/ipykernel_9676/941098153.py:8: UserWarning: [W008] Evaluating Doc.similarity based on empty vectors.
  similarity = test_doc1.similarity(test_doc2)
```

Statistical models are useful if your application needs to be able to generalize based on a few examples.Similarly, you can predict dependency labels to find subject/object relationships. Rule-based approaches on the other hand come in handy if there's a more or less finite number of instances you want to find. For example, all countries or cities of the world, drug names or even dog breeds.

To see the names of the pipeline components present in the current nlp object, you can use the nlp.pipe_names attribute.

For a list of component name and component function tuples, you can use the nlp.pipeline attribute.

The component functions are the functions applied to the doc to process it and set attributes – for example, part-of-speech tags or named entities.

```
nlp = spacy.load("nl_core_news_lg")
print(nlp.pipe_names)

['tok2vec', 'morphologizer', 'tagger', 'parser', 'lemmatizer', 'attribute_ruler', 'ner']

for component in nlp.pipeline:
    print(component)
```

```
('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec object at 0x1366c6940>)
('morphologizer', <spacy.pipeline.morphologizer.Morphologizer object
at 0x136153b20>)
('tagger', <spacy.pipeline.tagger.Tagger object at 0x116c31160>)
('parser', <spacy.pipeline.dep_parser.DependencyParser object at
0x136356190>)
('lemmatizer', <spacy.pipeline.edit_tree_lemmatizer.EditTreeLemmatizer
object at 0x1366cc820>)
('attribute_ruler', <spacy.pipeline.attributeruler.AttributeRuler
object at 0x17fb2e4c0>)
('ner', <spacy.pipeline.ner.EntityRecognizer object at 0x136b67e40>)
```

```
from spacy.language import Language

# Define a custom component
@Language.component("custom_component")
def custom_component_function(doc):
    # Print the doc's length
    print("Doc length:", len(doc))
    # Return the doc object
    return doc

# Add the component first in the pipeline
nlp.add_pipe("custom_component", first=True)
```

```
<function __main__.custom_component_function(doc)>
```

Custom components are great for adding custom values to documents, tokens and spans, and customizing the doc.ents.

```
nlp = spacy.load("nl_core_news_lg")
import spacy
from spacy.language import Language
from spacy.matcher import PhraseMatcher
from spacy.tokens import Span


dieren = ["wolf", "geitje", "moeder", "kudde", "schildpad", "eenden",
"Kat", "muizen", "haas", "vogels", "kraai", "leeuw", "muis", "vos",
"druiven", "Aap", "Kameel", "Trauben", "Stadtmaus", "Landmaus",
"Schaf", "Knochen", "Kranich", "Löwe", "Mäuschen"]
animal_patterns = list(nlp.pipe(dieren))
print("animal_patterns:", animal_patterns)
matcher = PhraseMatcher(nlp.vocab)
matcher.add("DIER", animal_patterns)

# Define the custom component
@Language.component("animal_component")
```

```
def animal_component_function(fabel4):
    # Apply the matcher to the doc
    matches = matcher(doc)
    # Create a Span for each match and assign the label "DIRE"
    spans = [Span(doc, start, end, label="DIERS") for match_id, start,
end in matches]
    # Overwrite the doc.ents with the matched spans
    doc.ents = spans
    return doc


# Add the component to the pipeline after the "ner" component
nlp.add_pipe("animal_component", after="ner")
print(nlp.pipe_names)

# Process the text and print the text and label for the doc.ents
print([(ent.text, ent.label_) for ent in fabel4.ents])
```

```
animal_patterns: [wolf, geitje, moeder, kudde, schildpad, eenden, Kat,
muizen, haas, vogels, kraai, leeuw, muis, vos, druiven, Aap, Kameel,
Trauben, Stadtmaus, Landmaus, Schaf, Knochen, Kranich, Löwe, Mäuschen]
['tok2vec', 'morphologizer', 'tagger', 'parser', 'lemmatizer',
'attribute_ruler', 'ner', 'animal_component']
[('Veldmuis', 'FAC'), ('dat.', 'GPE'), ('Stadsmuis', 'FAC'),
('Veldmuis', 'FAC'), ('volgende morgen', 'DATE'), ('Veldmuis', 'FAC'),
('volgende morgen', 'DATE'), ('Stadsmuis', 'FAC'), ('Veldmuis',
'FAC'), ('Veldmuis', 'FAC'), ('Veldmuis', 'FAC'), ('Kat',
'WORK_OF_ART'), ('Doodsbang', 'ORG'), ('Veldmuis', 'FAC'),
('Stadsmuis', 'FAC'), ('nederig', 'PERSON')]
```

```
displacy.render(fabel4, style="ent")
```

```
<IPython.core.display.HTML object>
```

it's tagging the mice as organizations locatins and events, To improve the accuracy of entity recognition for animals, I will consider training a custom entity recognition model on a labeled dataset that includes animal names. This would allow to specifically train the model to recognize animal entities and assign them the correct label.

```
#To see the available entity labels for the Dutch model in your
version of spaCy: (to be checked*****)
nlp = spacy.load("nl_core_news_lg")
print(nlp.pipe_labels["ner"])
```

```
['CARDINAL', 'DATE', 'EVENT', 'FAC', 'GPE', 'LANGUAGE', 'LAW', 'LOC',
'MONEY', 'NORP', 'ORDINAL', 'ORG', 'PERCENT', 'PERSON', 'PRODUCT',
'QUANTITY', 'TIME', 'WORK_OF_ART']
```

```
for ent in fabel1.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
Alstublieft 1004 1015 ORG
```

```
print("Noun phrases:", [chunk.text for chunk in fabel1.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ ==
"VERB"])

#Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

```
Noun phrases: ['een klein geitje', 'dat hoorntjes', 'hij', 'Hij', 'de
wei', 'zijn moeder', 'een grote kudde', 'geiten', 'hij', 'Elke avond',
'de geiten', 'hun stal', 'een avond', 'het klein geitje', 'de wei',
'Zijn moeder', 'huis', 'hij', 'het malse gras', 'hij', 'een tijdje',
'hij', 'zijn moeder en de andere geiten al', 'huis', 'Hij', 'De zon',
'lange schaduwen', 'de grond', 'Een koude wind', 'akelige geluiden',
'het gras', 'de bomen', 'Het geitje', 'de verschrikkelijke wolf',
'Hij', 'de weide', 'zijn moeder', 'hij', 'hij', 'een groepje',
'bomen', 'de wolf', 'Het geitje', 'het', 'de wolf', 'mijnheer', 'hij',
'Ik', 'u', 'uw blokfluit', 'een liedje', 'ik', 'ik', 'De wolf', 'een
leuk idee', 'een liedje', 'hij', 'het geitje', 'Hij', 'zijn
blokfluit', 'zijn mond', 'een liedje', 'Het geitje', 'het geluid', 'de
blokfluit', 'de honden', 'die', 'de geiten', 'Ze', 'het liedje', 'de
wolf', 'de wei', 'De wolf', 'zijn liedje', 'de honden', 'hij', 'hij',
'plaats', 'een liedje', 'hij', 'het geitje', 'je', 'je']
Verbs: []
```

```
fabel1 = nlp("Er was eens een klein geitje dat hoorntjes begon te
krijgen en daarom dacht dat hij nu al een grote geit was. Hij liep in
de wei, samen met zijn moeder en een grote kudde geiten, en zei tegen
iedereen dat hij nu wel voor zichzelf kon zorgen. Elke avond gingen de
geiten naar hun stal om er te slapen. Op een avond bleef het klein
geitje op de wei staan.  Zijn moeder riep hem om mee naar huis te
gaan. Maar hij wilde niet luisteren en bleef knabbelen aan het malse
gras. Toen hij na een tijdje rondkeek zag hij dat zijn moeder en de
andere geiten al naar huis waren. Hij was helemaal alleen. De zon ging
onder en er kropen lange schaduwen over de grond. Een koude wind stak
op en maakte akelige geluiden in het gras en in de bomen. Het geitje
rilde toen het dacht aan de verschrikkelijke wolf. Hij liep snel over
de weide en begon te roepen op zijn moeder. Maar hij was nog niet
halfweg toen hij, naast een groepje bomen, de wolf zag staan! Het
geitje was bang want het wist dat de wolf hem zou opeten.
“Alstublieft, mijnheer de wolf” zei hij bevend “Ik weet dat u mij gaat
opeten. Maar speel eerst op uw blokfluit een liedje voor mij, want ik
wil dansen en vrolijk zijn, zolang als ik kan.” De wolf vond het een
leuk idee om eerst een liedje te spelen vooraleer hij het geitje zou
opeten.  Hij zette zijn blokfluit aan zijn mond en speelde een liedje.
Het geitje begon vrolijk te dansen en rond te springen. Maar het
geluid van de blokfluit werd gehoord door de honden die de geiten
beschermen. Ze herkenden het liedje van de wolf en begonnen heel hard
```

```
naar de wei te lopen. De wolf hield ineens op met zijn liedje en liep
snel weg.Terwijl de honden achter hem zaten was hij boos op zichzelf
omdat hij zo dom was geweest. In plaats van eerst een liedje te spelen
had hij beter het geitje onmiddellijk opgegeten. Laat je nooit
afleiden als je iets wilt bereiken.")

fabel1.ents = [
    Span(fabel1, 0, 1, label="WEBSITE"),
    Span(fabel1, 3, 4, label="WEBSITE"),
]

# Import global classes
from spacy.tokens import Doc, Token, Span

# Set extensions on the Doc, Token and Span
Doc.set_extension("title", default=None)
Token.set_extension("is_color", default=False)
Span.set_extension("has_color", default=False)
```

Extension attribute types Attribute extensions Property extensions Method extensions

```
#processing large volumes of text
# use nlp.make_doc to turn a text into a Doc object
# doc = nlp.make_doc("Hello World!")
docs = list(nlp.pipe(text))
```

use nlp.select_pipes(disable=["tagger, "parser"]) doc = nlp(text) print(doc.ents)

```
#displacy.render(docs, style="ent")

#nlp = spacy.load("nl_core_news_lg")
# Process the texts and print the adjectives
#for doc in nlp.pipe(docs):
    #print([token.text for token in doc if token.pos_ == "ADJ"])

# Process the texts and print the entitities
#docus = list(nlp.pipe(docs))
#entities = [docus.ent for docu in docus]
#print(*entities)

dieren = ["wolf", "geitje", "moeder", "kudde", "schildpad", "eenden",
"Kat", "muizen", "haas", "vogels", "kraai", "leeuw", "muis", "vos",
"druiven", "Aap", "Kameel", "Trauben", "Stadtmaus", "Landmaus",
"Schaf", "Knochen", "Kranich", "Löwe", "Mäuschen"]
# Create a list of patters for the PhraseMatcher
patterns = list(nlp.pipe(dieren))
```