The repository for this class can be found here on GitHub.

Exercise 2.1 Playing around with the NLTK Book Collection

First, we import the module and download the books. The outputs are shown as comments.

```
import re

# Started with the following to load the book corpus
# in the module's data directory (only once)
import nltk
nltk.download()
nltk.download('punkt_tab')
from nltk import book as books

""" outputs
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
"""
```

Then, we start playing the methods and discovering the parameters of the book module (imported as "books" above). As we can see, some methods are more useful than others. The `sents` method for instance doesn't do anything but printing the first sentence from each book.

```
print(books.text1.name)  # Moby Dick by Herman Melville 1851
books.sents()
"""
sent1: Call me Ishmael .
sent2: The family of Dashwood had long been settled in Sussex .
sent3: In the beginning God created the heaven and the earth .
sent4: Fellow - Citizens of the Senate and of the House of Representatives :
sent5: I have a problem with people PMing me to lol JOIN
sent6: SCENE 1 : [ wind ] [ clop clop clop ] KING ARTHUR : Whoa there !
sent7: Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
sent8: 25 SEXY MALE , seeks attrac older single lady , for discreet encounters .
sent9: THE suburb of Saffron Park lay on the sunset side of London , as red and ragged as a cloud of
sunset .
"""

books.text3.concordance("thou")
"""
Displaying 25 of 284 matches:
saying , Of every tree of the garden thou mayest freely e But of the tree of t
 of the knowledge of good and evil , thou shalt not eat of for in the day that
shalt not eat of for in the day that thou eatest thereof thou shalt surely die
 in the day that thou eatest thereof thou shalt surely die . And the LORD God
Adam , and said unto him , Where art thou ? And he said , I heard thy voice in
```

```
f . And he said , Who told thee that thou wast naked ? Hast thou eaten of the
old thee that thou wast naked ? Hast thou eaten of the tree , whereof I comman
tree , whereof I commanded thee that thou shouldest not eat ? And the man said
 ? And the man said , The woman whom thou gavest to be with me , she gave me o
d unto the woman , What is this that thou hast done ? And the woman said , The
 God said unto the serpent , Because thou hast done this , thou art cursed abo
pent , Because thou hast done this , thou art cursed above all cattle , and ab
 of the field ; upon thy belly shalt thou go , and dust shalt thou eat all the
belly shalt thou go , and dust shalt thou eat all the days of thy li And I wil
eed ; it shall bruise thy head , and thou shalt bruise his heel . Unto the wom
orrow and thy conception ; in sorrow thou shalt bring forth children ; and thy
ee . And unto Adam he said , Because thou hast hearkened unto the voice of thy
of which I commanded thee , saying , Thou shalt not eat of cursed is the groun
round for thy sake ; in sorrow shalt thou eat of it all the days of thy life ;
s shall it bring forth to thee ; and thou shalt eat the herb of the field ; In
eld ; In the sweat of thy face shalt thou eat bread , till thou return unto th
thy face shalt thou eat bread , till thou return unto the ground ; for out of
unto the ground ; for out of it wast thou tak for dust thou art , and unto dus
for out of it wast thou tak for dust thou art , and unto dust shalt thou retur
 dust thou art , and unto dust shalt thou return . And Adam called his wife ' """

print(books.text6.generate())
"""
SOLDIER # 1 : Until you come and get a bit . # 2 : Hic ! ARTHUR : How
can we come up and have a plan , sir . roar ] MAYNARD : Armaments ,
Chapter Two , verses Nine to Twenty - one . Heh . DINGO : Yes , yes ,
you ain ' t . , great . Ni ! who , when he seemed about to recover ,
suddenly felt the icy hand of death upon him . No SCENE 19 : ARTHUR :
Walk away . squeak ] [ howl ] [ creak
SOLDIER # 1 : Until you come and get a bit . # 2 : Hic ! ARTHUR : How
can we come up and have a plan , sir . roar ] MAYNARD : Armaments ,
Chapter Two , verses Nine to Twenty - one . Heh . DINGO : Yes , yes ,
you ain ' t . , great . Ni ! who , when he seemed about to recover ,
suddenly felt the icy hand of death upon him . No SCENE 19 : ARTHUR :
Walk away . squeak ] [ howl ] [ creak
"""
```

Using the shell I could discover the other methods of the texts modules.

```
# example of methods coming with a text
"""
book.text4.collocation_list(  book.text4.concordance_list(  book.text4.generate(
book.text4.readability(
book.text4.collocations(      book.text4.count(             book.text4.index(
book.text4.similar(
book.text4.common_contexts(   book.text4.dispersion_plot(   book.text4.name
book.text4.tokens
book.text4.concordance(       book.text4.findall(           book.text4.plot(
book.text4.vocab()
>>> book.text4.
 """
```

Finding the first 100 words of 'Monty Python and the Holy Grail' depends on what we define as a word. In the code bellow, we admit two definitions, a word as a token and a word as a type.

```
types = set()
for word in books.text6:
    if len(types) == 100:
        break
    # Include in the set if it contains a letter
    types |= {re.sub(r"^[^a-zA-Z]+$", "", word) or None}
```

```
print("Analyse of the Monty Python and the Holy Grail:")
print(
    f"""First 100 tokens: {books.text6[:100]}
First 100 wordtypes: {types}
"""
)
```

```
"""
Analyse of the Monty Python and the Holy Grail:

First 100 tokens: ['SCENE', '1', ':', '[', 'wind', ']', '[', 'clop', 'clop', 'clop', ']', 'KING',
'ARTHUR', ':', 'Whoa', 'there', '!', '[', 'clop', 'clop', 'clop', ']', 'SOLDIER', '#', '1', ':',
'Halt', '!', 'Who', 'goes', 'there', '?', 'ARTHUR', ':', 'It', 'is', 'I', ',', 'Arthur', ',', 'son',
'of', 'Uther', 'Pendragon', ',', 'from', 'the', 'castle', 'of', 'Camelot', '.', 'King', 'of', 'the',
'Britons', ',', 'defeator', 'of', 'the', 'Saxons', ',', 'sovereign', 'of', 'all', 'England', '!',
'SOLDIER', '#', '1', ':', 'Pull', 'the', 'other', 'one', '!', 'ARTHUR', ':', 'I', 'am', ',', '...',
'and', 'this', 'is', 'my', 'trusty', 'servant', 'Patsy', '.', 'We', 'have', 'ridden', 'the',
'length', 'and', 'breadth', 'of', 'the', 'land', 'in']
First 100 types: First 100 types (including punctuation): {'is', 'England', 'a', 'Ridden', 've',
'join', 'snows', 'goes', 'winter', 'Whoa', 'tropical', 'Found', None, 'am', 'Where', 'son', 'We',
'will', 'found', 'King', 'Arthur', 'SCENE', 'knights', 'Camelot', 'me', 'at', 'lord', 'and', 'em',
'using', 'together', 'SOLDIER', 'two', 'coconut', 'halves', 'must', 'coconuts', 'kingdom', 'speak',
's', 'Yes', 'Mercea', 'It', 'defeator', 'court', 'got', 'land', 'clop', 'other', 'covered', 'them',
'I', 'breadth', 'from', 'You', 'empty', 'since', 'horse', 'there', 'ARTHUR', 're', 'In', 'Uther',
'servant', 'Pendragon', 'castle', 'sovereign', 'wind', 'What', 'all', 'you', 'through', 'Pull',
'one', 'in', 'Saxons', 'bangin', 'my', 'this', 'So', 'Halt', 'your', 'The', 'KING', 'd', 'have',
'master', 'ridden', 'Patsy', 'with', 'who', 'Who', 'the', 'length', 'search', 'Britons', 'get', 'on',
'trusty', 'of'}
"""
```

Exercise 2.2 Introducing Regular Expressions

In this section, we will learn new English words that we can list based on different criteria using regular expressions.

```
import re
import nltk

words = nltk.corpus.words.words("en")

only_vowel_words = [word for word in words if re.match("^[aeiouAEIOU]{3}$", word)]
print(only_vowel_words)
# ['iao', 'oii']

vcv = [
    word for word in words if re.match("^[aeiouAEIOU][b-df-hj-np-tv-z][aeiou]$", word)
]
print(vcv)
# ['aba', 'Abe', 'Abo', 'Abu', 'abu', 'ace', 'Ada', 'Ade', 'ade', 'ado', 'aga', 'age', 'ago', 'aha',
'aho', 'ahu', 'Aka', 'aka', 'ake', 'ako', 'aku', 'ala', 'ale', 'alo', 'ama', 'ame', 'Ami', 'ami',
'Ana', 'ana', 'ani', 'apa', 'ape', 'ara', 'are', 'Aro', 'aru', 'Asa', 'ase', 'Ata', 'ate', 'Ati',
'ava', 'Ave', 'ave', 'avo', 'awa', 'awe', 'axe', 'aye', 'ayu', 'azo', 'Edo', 'ego', 'eke', 'Eli',
'eme', 'emu', 'era', 'ere', 'eta', 'Eva', 'Eve', 'eve', 'Ewe', 'ewe', 'eye', 'iba', 'Ibo', 'ice',
'Ida', 'ide', 'Ido', 'ife', 'ihi', 'Ijo', 'Ike', 'Ila', 'Ima', 'imi', 'imu', 'Ino', 'Ira', 'ire',
'iso', 'Ita', 'Ito', 'iva', 'iwa', 'iyo', 'obe', 'obi', 'oda', 'ode', 'Ofo', 'oho', 'oka', 'oki',
'Ole', 'Ona', 'ona', 'one', 'ope', 'ora', 'ore', 'ose', 'Oto', 'Ova', 'ova', 'owe', 'ubi', 'Uca',
'Udi', 'udo', 'uji', 'uke', 'ula', 'ule', 'ulu', 'ume', 'umu', 'Una', 'upo', 'ura', 'ure', 'Uri',
'Uro', 'Uru', 'use', 'Uta', 'uta', 'Ute', 'utu', 'uva']

vcvv = [
    word
    for word in words
```

```
    if re.match("^[aeiouAEIOU][b-df-hj-np-tv-z][aeiou]{2}$", word)
]
print(vcvv)
# ['Abie', 'Adai', 'Agao', 'Agau', 'agee', 'agio', 'agua', 'ague', 'akee', 'akia', 'Alea', 'alee',
'aloe', 'Amia', 'anoa', 'apii', 'apio', 'aqua', 'aquo', 'area', 'aria', 'arui', 'awee', 'Eboe',
'eboe', 'edea', 'eheu', 'ejoo', 'Ekoi', 'Elia', 'epee', 'eria', 'Erie', 'etua', 'etui', 'Evea',
'evoe', 'idea', 'ilia', 'Inia', 'Itea', 'Ixia', 'oboe', 'ogee', 'ohia', 'Ohio', 'okee', 'okia',
'Okie', 'Olea', 'oleo', 'olio', 'omao', 'oxea', 'Ubii', 'Ulua', 'ulua', 'unau', 'unie', 'Unio',
'unio', 'urao', 'urea', 'Uria', 'usee', 'utai', 'uvea']


vcvvv = [
    word
    for word in words
    if re.match("^[aeiouAEIOU][b-df-hj-np-tv-z][aeiou]{3}$", word)
]
print(vcvvv)
# ['adieu', 'Araua', 'Arioi', 'iliau', 'Umaua']

len_7_adehins = [word for word in words if re.match(r"^[adehins]{7}$", word)]
print(len_7_adehins)
# ['addenda', 'adenase', 'adenine', 'ahaaina', 'anidian', 'aniseed', 'asinine', 'asshead', 'daisied',
'danaide', 'danaine', 'dasheen', 'deadish', 'deaness', 'deedeed', 'disdain', 'disease', 'enshade',
'hashish', 'hennish', 'insense', 'nandine', 'saddish', 'sadness', 'seaside', 'shadine', 'shedded']

textonyms = [word for word in words if re.match("^[defDEF][ghi][jkl][mno]$", word)]
print(textonyms)
# ['dilo', 'film', 'filo']
```

2.3 Tagging Parts of Speech (POS)

```
import re
import nltk


def extract_first_noun_phrase(sentence: str) -> str:
    # Tokenize the text into words
    tokens = nltk.word_tokenize(sentence)

    first_noun_phrase = []
    for token in nltk.pos_tag(tokens):
        if (
            len(first_noun_phrase) > 0
            and "NN" == first_noun_phrase[-1][1][:2]
            and (token[1][:2] != "NN" and token[1][:2] != "IN" and token[1][:2] != "PR")
        ):
            # end the loop if the first noun phrase has ended
            break
        elif re.match(r"^(DT|JJ|IN|NNS?|PR.*)$", token[1]):
            # extend the phrase as long as a POS tag is a valid part of a NP
            first_noun_phrase.append(token)

    return " ".join([word for word, tag in first_noun_phrase])


# testing the code with sample sentences
sample_sentences = [
    "Should all good men to come to the aid of their party?",
    "And now is the time for all good men to come to the aid of their party",
    "and for all good men to come to the aid of their party.",
    "Come to the aid of your party.",
    "Your party needs you.",
]

for sentence in sample_sentences:
```

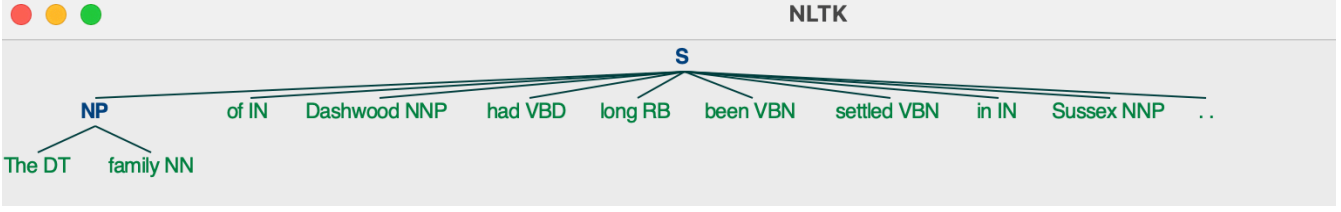
```
    print(f"Sentence: {sentence}")
    print(f"First noun phrase: {extract_first_noun_phrase(sentence)}")
    print()

"""
Sentence: Should all good men to come to the aid of their party?
First noun phrase: all good men

Sentence: And now is the time for all good men to come to the aid of their party
First noun phrase: the time for all good men

Sentence: and for all good men to come to the aid of their party.
First noun phrase: for all good men

Sentence: Come to the aid of your party.
First noun phrase: the aid of your party

Sentence: Your party needs you.
First noun phrase: Your party needs you
"""
```

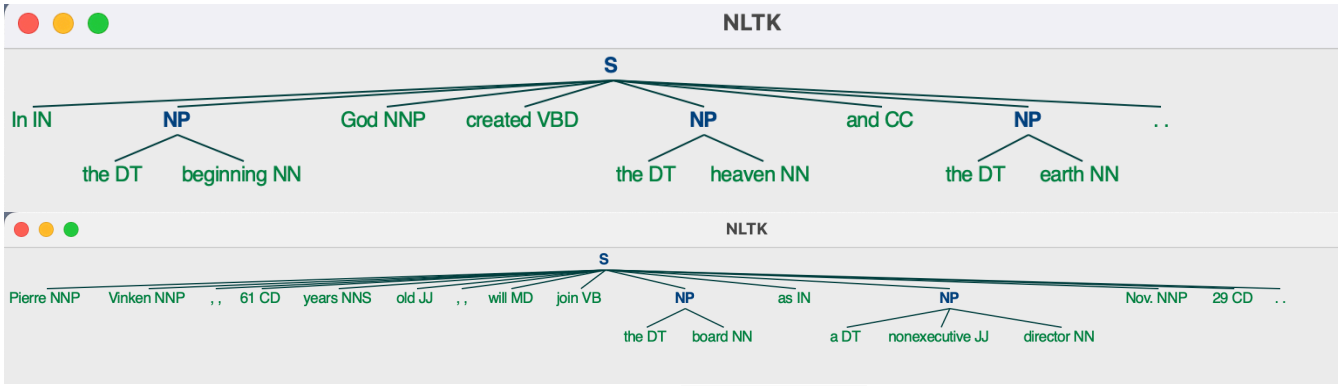
In this program I attempted to use regular expressions to extract the first noun phrase from a given sentence. To do so I had to extend the definition provided in the assignment to take more part of speech (POS) tags into account. Especially, the plurial nouns, `NNS`, the personal pronouns (which are between determinants and adjectives), and the tag "TO" which symbolizes a preposition.
It is important to point out that no matter how complete the function we use to extract the noun phrase can be, it is always dependant on the tagging that is made by the NLTK `pos_tag` module. This module might fail to recognise properly the the parts of a sentence. In the last sample sentence, the toke "needs" is parsed as a plurial noun `NNS`, although it is in reality a conjugated verb. Since NLTK did not categorise the verb properly, my function also integrated the following pronoun as a part of the noun phrase, despite English syntax not allowing a noun phrase to end by a pronoun... I still keep this output for reference.

2.4 Making syntactic trees

```
import nltk
from nltk import book

np_grammar = "NP: {<DT>?<JJ>*<NN>+}"

np_parser = nltk.RegexpParser(np_grammar)

sentences = [
    nltk.pos_tag(sentence) for sentence in [book.sent2, book.sent3, book.sent5]
]
res = ""
for sentence in sentences:
    print(" ".join([token for token, tag in sentence]))
    result = np_parser.parse(sentence)
    print(result)
    result.pretty_print()
    result.draw()

""" output
The family of Dashwood had long been settled in Sussex .
(S
  (NP The/DT family/NN)
  of/IN
  Dashwood/NNP
  had/VBD
  long/RB
  been/VBN
  settled/VBN
```

```
  in/IN
  Sussex/NNP
  ./.)
                                                S
   _____________________________________________|_____________________________________
  |        |          |       |       |          |        |       |       |          NP
  |        |          |       |       |          |        |       |       |     _____|______
of/IN Dashwood/NNP had/VBD long/RB been/VBN settled/VBN in/IN Sussex/NNP ./. The/DT     family/NN

2025-02-26 15:51:32.042 Python[6713:91178] +[IMKClient subclass]: chose IMKClient_Modern
2025-02-26 15:51:32.042 Python[6713:91178] +[IMKInputSession subclass]: chose IMKInputSession_Modern
In the beginning God created the heaven and the earth .
(S
  In/IN
  (NP the/DT beginning/NN)
  God/NNP
  created/VBD
  (NP the/DT heaven/NN)
  and/CC
  (NP the/DT earth/NN)
  ./.)
                                      S
   ___________________________________|_____________________________________________
  |      |         |        |     |          NP                    NP                  NP
  |      |         |        |     |     _____|_______          _____|_______     _____|_____
In/IN God/NNP created/VBD and/CC ./. the/DT     beginning/NN the/DT     heaven/NN the/DT     earth/NN

Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
(S
  Pierre/NNP
  Vinken/NNP
  ,/,
  61/CD
  years/NNS
  old/JJ
  ,/,
  will/MD
  join/VB
  (NP the/DT board/NN)
  as/IN
  (NP a/DT nonexecutive/JJ director/NN)
  Nov./NNP
  29/CD
  ./.)
                                                        S
_______________________________________________________________|_____________________________________
_____________________
    |          |       |    |       |       |     |     |       |      |      |       |    |
NP                        NP
    |          |       |    |       |       |     |     |       |      |      |       |    |
_____|_____        __________|_____________
Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB as/IN Nov./NNP 29/CD ./. the/DT
board/NN a/DT nonexecutive/JJ director/NN
"""

"""
```





As I initially struggled to get tkinter working, I added a `pretty_print` of the tree to render them in the terminal. Those pretty print are not ideal because they show the noun phrase at the end of the tree. One important thing to notice about the chunks made by the simple grammar `"NP: {<DT>?<JJ>*<NN>+}"` is that they will not take into a plurial nouns (NNS) nor proper nouns (NNP) like "God" in the second sentence above. This underscore the importance of intensive testing in the field of NLP, as rules are not always so straightforward as one might think.

2.5 Bluild an advanced chunker

```
import nltk

np_grammar = r"""
    NP: {<POS>?(<DT>?<PRP\$>?<\#>?<CC>?<JJS?R?>*<VBG>*<NNP?S?|(CD)>+)+}
"""

np_parser = nltk.RegexpParser(np_grammar)

phrases = """another/DT sharp/JJ dive/NN
trade/NN figures/NNS
any/DT new/JJ policy/NN measures/NNS
earlier/JJR stages/NNS
Panamanian/JJ dictator/NN Manuel/NNP Noriega/NNP
his/PRP$ Mansion/NNP House/NNP speech/NN
the/DT price/NN cutting/VBG
3/CD %/NN to/TO 4/CD %/NN
more/JJR than/IN 10/CD %/NN
the/DT fastest/JJS developing/VBG trends/NNS
's/POS skill/NN""".split(
    "\n"
)
# First, we prepare the samples in order to process them with nltk
phrases = [[tuple(word.split("/")) for word in phrase.split(" ")] for phrase in phrases]

# using the chunkparser to reach an F-score of 75%
# Precision: 81% and Recall 70%
# nltk.app.chunkparser()

for phrase in phrases:
    parsed = np_parser.parse(phrase)
    print(parsed)

"""
(S (NP another/DT sharp/JJ dive/NN))
(S (NP trade/NN figures/NNS))
(S (NP any/DT new/JJ policy/NN measures/NNS))
(S (NP earlier/JJR stages/NNS))
(S (NP Panamanian/JJ dictator/NN Manuel/NNP Noriega/NNP))
(S (NP his/PRP$ Mansion/NNP House/NNP speech/NN))
(S (NP the/DT price/NN) cutting/VBG)
(S (NP 3/CD %/NN) to/TO (NP 4/CD %/NN))
(S more/JJR than/IN (NP 10/CD %/NN))
```

```
(S (NP the/DT fastest/JJS developing/VBG trends/NNS))
(S (NP 's/POS skill/NN))
"""
```

In this section I managed to build a grammar with an F-score of 75%. It was hard to go higher, but also unnecessary for the short sample set given in the assignment. It is noteworthy that there are some mistakes in these phrases. "more than 10%" technically does not contain any noun phrase, because 10% is a quantifier phrase, it has no proper NP head. But based on the provided data, tagging "%" as a noun, and not a quantifier, the phrase "10%" is parsed as a noun by my grammar. This relates to the problem of the dependance on correct tagging that I highlighted in part 3. This shows how bad data can corrupt the whole pipeline in a cascading effect.

2.6 Named Entities

```
import nltk

sentences = """
Throughout history, never has the USA seen such an incompetent administration.
Throughout history, never has the USA seen such an Incompetent Administration.
The 47th president Donald Duck's impredictability is only surpassed by his amateurism.
The only thing the king of D.C. is ruining faster than the American credibility is the American
economy.
Since January, the stock exchange market seems to reach new low every hour.
Especially bad is the value of Musk's companies shares, with Tesla share trading at 281$ at 8 p.m.
today.
The White house is in trouble, because the USA biggest hazard is already inside.
"""

cleaned_sentences = [
    nltk.pos_tag(nltk.word_tokenize(sentence))
    for sentence in sentences.strip().split("\n")
]

for sentence in cleaned_sentences:
    print(nltk.ne_chunk(sentence))

"""
(S
  Throughout/IN
  history/NN
  ,/,
  never/RB
  has/VBZ
  the/DT
  (ORGANIZATION USA/NNP)
  seen/VBN
  such/PDT
  an/DT
  incompetent/JJ
  administration/NN
  ./.)
(S
  Throughout/IN
  history/NN
  ,/,
  never/RB
  has/VBZ
  the/DT
  (ORGANIZATION USA/NNP)
  seen/VBN
  such/PDT
  an/DT
  incompetent/JJ
```

```
  (ORGANIZATION Administration/NN)
  ./.)
(S
  Throughout/IN
  history/NN
  ,/,
  never/RB
  has/VBZ
  the/DT
  (ORGANIZATION USA/NNP)
  seen/VBN
  such/PDT
  an/DT
  Incompetent/NNP
  administration/NN
  ./.)
(S
  The/DT
  47th/CD
  president/NN
  (PERSON Donald/NNP Duck/NNP)
  's/POS
  impredictability/NN
  is/VBZ
  only/RB
  surpassed/VBN
  by/IN
  his/PRP$
  amateurism/NN
  ./.)
(S
  The/DT
  only/JJ
  thing/NN
  the/DT
  king/NN
  of/IN
  (GPE D.C./NNP)
  is/VBZ
  ruining/VBG
  faster/RBR
  than/IN
  the/DT
  (GPE American/JJ)
  credibility/NN
  is/VBZ
  the/DT
  (GPE American/JJ)
  economy/NN
  ./.)
(S
  Since/IN
  January/NNP
  ,/,
  the/DT
  stock/NN
  exchange/NN
  market/NN
  seems/VBZ
  to/TO
  reach/VB
  new/JJ
  low/JJ
  every/DT
  hour/NN
  ./.)
(S
```

```
  Especially/RB
  bad/JJ
  is/VBZ
  the/DT
  value/NN
  of/IN
  (PERSON Musk/NNP)
  's/POS
  companies/NNS
  shares/NNS
  ,/,
  with/IN
  (PERSON Tesla/NNP)
  share/NN
  trading/NN
  at/IN
  281/CD
  $/$
  at/IN
  8/CD
  p.m./NN
  today/NN
  ./.)
(S
  The/DT
  (FACILITY White/NNP)
  house/NN
  is/VBZ
  in/IN
  trouble/NN
  ,/,
  because/IN
  the/DT
  (ORGANIZATION USA/NNP)
  biggest/JJS
  hazard/NN
  is/VBZ
  already/RB
  inside/RB
  ./.)
"""
```

I tried to include all the different named entities (NE) listed in the assignment in my sample sententences to see how the system would categorize them. Some of the words used are voluntarily abiguous, such as the word "administration" in the first sentence. If I turn the first letter of this word to uppercase ("Administration"), the system parse it as a NE (namely, `Organisation`). But initial uppercase does not turn any word into a named entity, see the third sentence with an uppercase "Incompetent" not being parsed as a NE.
But the system is not "intelligent" enough to correct language mistakes. I found out that I misspelled the word "White House" in the last sentece, and only the word White was included as a NE. The system also failed to recognise "Tesla" as an `Organisation`, and parsed it as a `Person` instead. "USA" is aparently an `Organisation` and not a `GPE` for the system, which is another mistake.
Overall, the system rather successfully recognise where the NE are located in the sentences. But this is just the first layer of understanding, the next layer of conceptualisation is a harder task and the system performs significantly worse when trying to categorise precisely what sort of NE they are facing. Nowaday, transformer-based architectures cracked this next level of abstraction by mixing the embedded value of the word with the value of the other elements of the surounding context.

2.7 Relation Extraction

```
import nltk
from nltk import ne_chunk, pos_tag, word_tokenize
```

```
import re
import requests
from bs4 import BeautifulSoup

# Extract the History section of the Artificial_intelligence" Wikipedia page
# Get Wikipedia page
url = "https://en.wikipedia.org/wiki/Bell_labs"
print("Loading the page content")
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

print("Parsing the page content")
# Find history section (starts after heading "History")
page = soup.find("div", {"id": "bodyContent"})

text_content = []

# Get all paragraphs
paragraphs = page.find_all("p")

# Extract text from each paragraph
for p in paragraphs:
    # Skip empty paragraphs and footnotes
    if p.text.strip() and not p.find_parent(class_="reference"):
        text_content.append(p.text)


# Join paragraphs and clean text
text = " ".join(text_content)
text = text.replace("\n", " ").replace("[edit]", "").strip()
# Remove citations [1], [2], etc.
text = (
    re.sub(r"\[\d+\]", "", text) + " The Bell Labs was located in New Jersey in 1970."
)

print("tokenizing, tagging and chunking")
sequences = [
    ne_chunk(pos_tag(word_tokenize(sentence + ". "))) for sentence in text.split(". ")
]

print("Start ectracting the relations")
rels = []
for sequence in sequences:
    new_relation = {"ORG": None, "isIn": False, "LOC": None}
    for chunk in sequence:
        if type(chunk) != tuple:
            if chunk.label() == "ORGANIZATION":
                new_relation["ORG"] = " ".join([leaf[0] for leaf in chunk])
            elif (
                (chunk.label() == "GPE" or chunk.label() == "LOCATION")
                and new_relation["ORG"] != None
                and new_relation["isIn"]
            ):
                new_relation["LOC"] = " ".join([leaf[0] for leaf in chunk])
                rels.append(new_relation)
                new_relation = {"ORG": None, "isIn": False, "LOC": None}

        elif new_relation["ORG"] is not None and chunk[1] == "IN" and chunk[0] == "in":
            new_relation["isIn"] = True
for rel in rels:
    print(f"[ORG: '{rel["ORG"]}'] 'in' [LOC: '{rel["LOC"]}'] ")

"""
Loading the page content
Parsing the page content
tokenizing, tagging and chunking
Start ectracting the relations
```

```
[ORG: 'Bell Telephone Laboratories'] 'in' [LOC: 'Western Electric']
[ORG: 'Volta Laboratory'] 'in' [LOC: 'Washington']
[ORG: 'Bell'] 'in' [LOC: 'Western Electric']
[ORG: 'Bell Laboratories'] 'in' [LOC: 'New Jersey']
[ORG: 'Bell Laboratories'] 'in' [LOC: 'New Jersey']
[ORG: 'Bell Lab'] 'in' [LOC: 'United States']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Bell Labs'] 'in' [LOC: 'Freehold']
[ORG: 'Bell Laboratories Fellow'] 'in' [LOC: 'United States']
[ORG: 'National Medal'] 'in' [LOC: 'Technology']
[ORG: 'Yokosuka Research Park'] 'in' [LOC: 'Yokosuka']
[ORG: 'Bell Labs Fellow Award'] 'in' [LOC: 'Network Architecture']
[ORG: 'Professional Services'] 'in' [LOC: 'Cable']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Bell Labs Fellow'] 'in' [LOC: 'North']
[ORG: 'Bell Labs'] 'in' [LOC: 'Information']
[ORG: 'Bell Labs'] 'in' [LOC: 'Tel Aviv']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Chemistry']
[ORG: 'Nobel Prize'] 'in' [LOC: 'Physics']
[ORG: 'Bell Labs'] 'in' [LOC: 'New Jersey']
[ORG: 'Bell Labs'] 'in' [LOC: 'New Jersey']
"""
```

In this snippet, I tried to extract the relations between locations and organisation from the wikipedia article of the Bell Labs (and one sentence that I used for testing the parsing code, as the whole thing take more than two minutes to run). We learn about sister organisations like the Volta laboratory and the Bell Labs Fellow, as well as the different locations where the labs used to be located.
One noteworthy phenomenon here is how Nobel prices get parsed as an organisation located in Physics, which is not a region of Sweden. For the same reasons as in the previous exercise, the title Nobel Prize in Physics is tagged and chunked inproperly by NLTK, which make this kind of error almost impossible to correct in subsequent layers. This shows how essential is the data preparation and a proper testing of the more basic building blocs of any NLP pipeline.