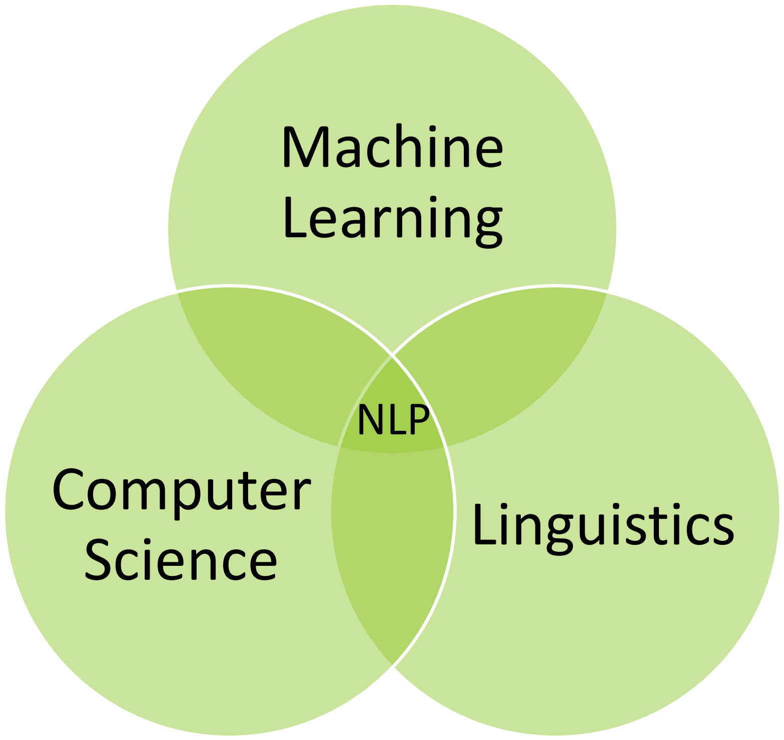
ENSF 612
Lecture - Natural Language Processing (NLP)
Basic Language Processing using Python

Gias Uddin, Assistant Professor
Electrical and Software Engineering,
University of Calgary
https://giasuddin.ca/

Topics

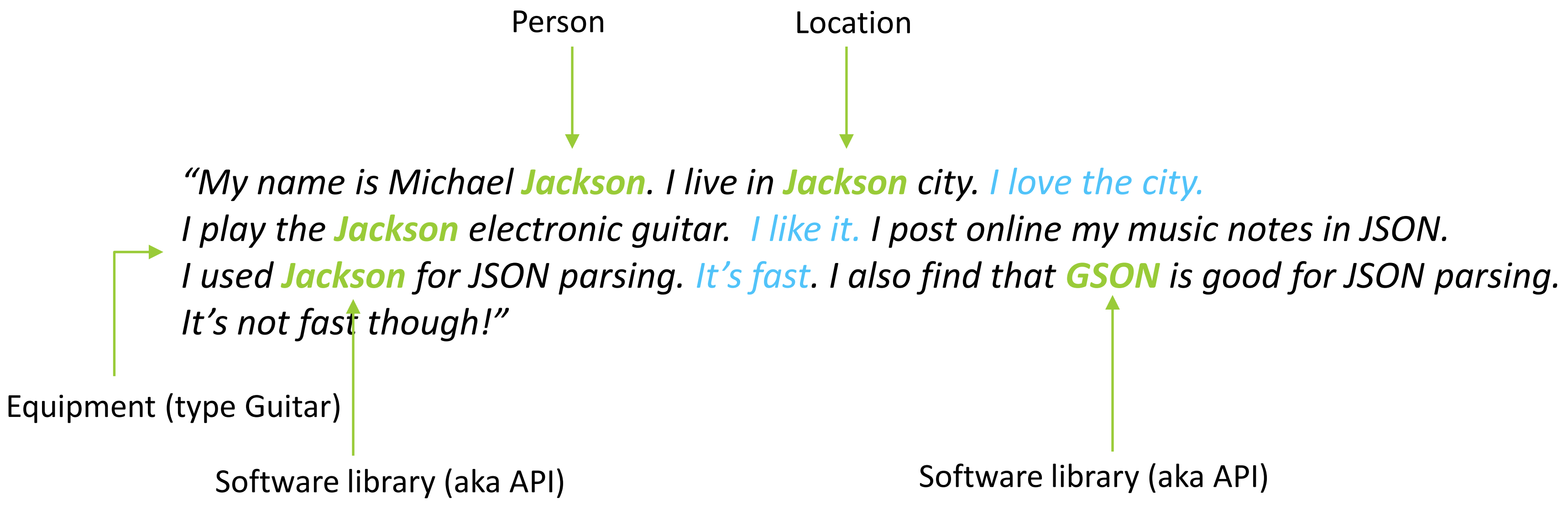
- What is NLP?
- Natural Language Toolkit (NLTK)
- Basic NLP tasks using NLTK

Natural Language Processing (NLP)



"NLP is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, how to program computers to process and analyze large amounts of natural language data." (Wikipedia)

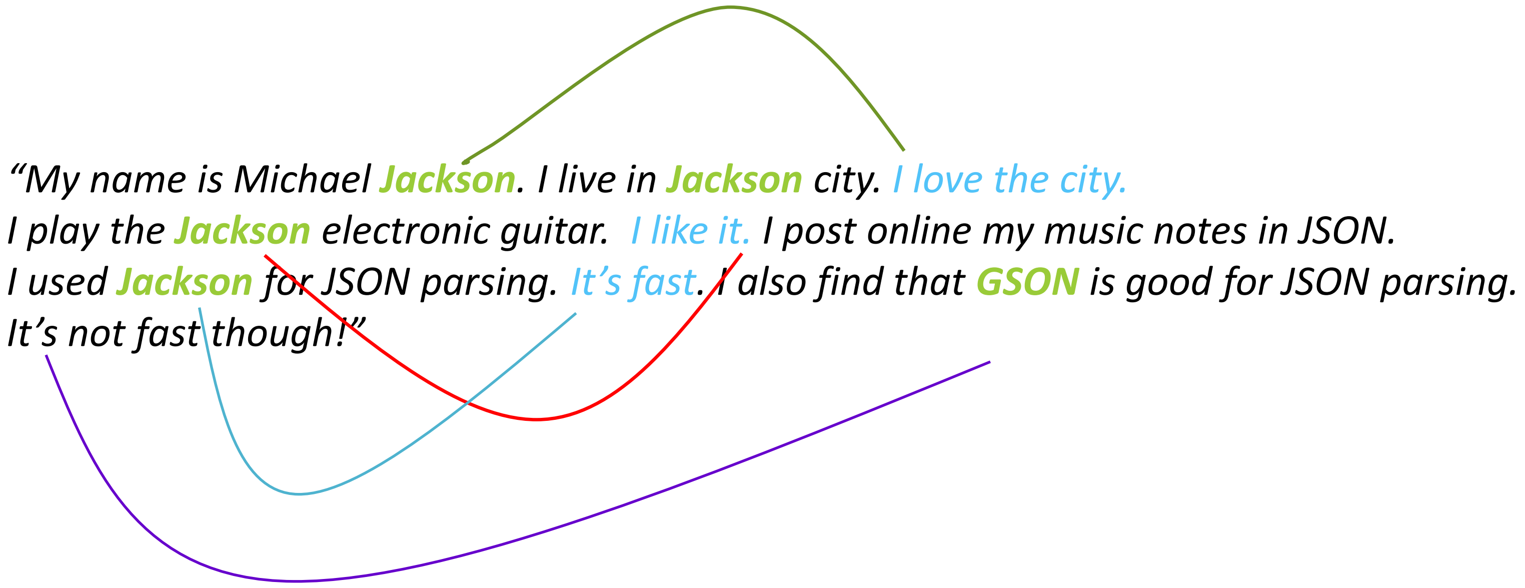
Some Basic Applications of NLP



*"My name is Michael **Jackson**. I live in **Jackson** city. I love the city.
I play the **Jackson** electronic guitar. I like it. I post online my music notes in JSON.
I used **Jackson** for JSON parsing. It's fast. I also find that **GSON** is good for JSON parsing.
It's not fast though!"*

1. How can we automatically detect entity names in natural language texts?
   - Named Entity Detection (NED)
2. How can we automatically resolve entity names and types in texts?
   - Named Entity Resolution (NER)

Some Basic Applications of NLP



*"My name is Michael **Jackson**. I live in **Jackson** city. I love the city.*
*I play the **Jackson** electronic guitar. I like it. I post online my music notes in JSON.*
*I used **Jackson** for JSON parsing. It's fast. I also find that **GSON** is good for JSON parsing.*
It's not fast though!"

1. How can we automatically associate a direct/indirect reference (e.g., pronouns) to a named entity in natural language texts?
   - Co-reference resolution

Domains where NLP can be applied

Many other applications of NLP

1. Fake news detection
2. Trust and bias in the expressed opinions
3. Intent classification (you are already introduced to it)
4. Text summarization
5. Automatic Question and Answer assistant
6. Disambiguate underlying context/meaning

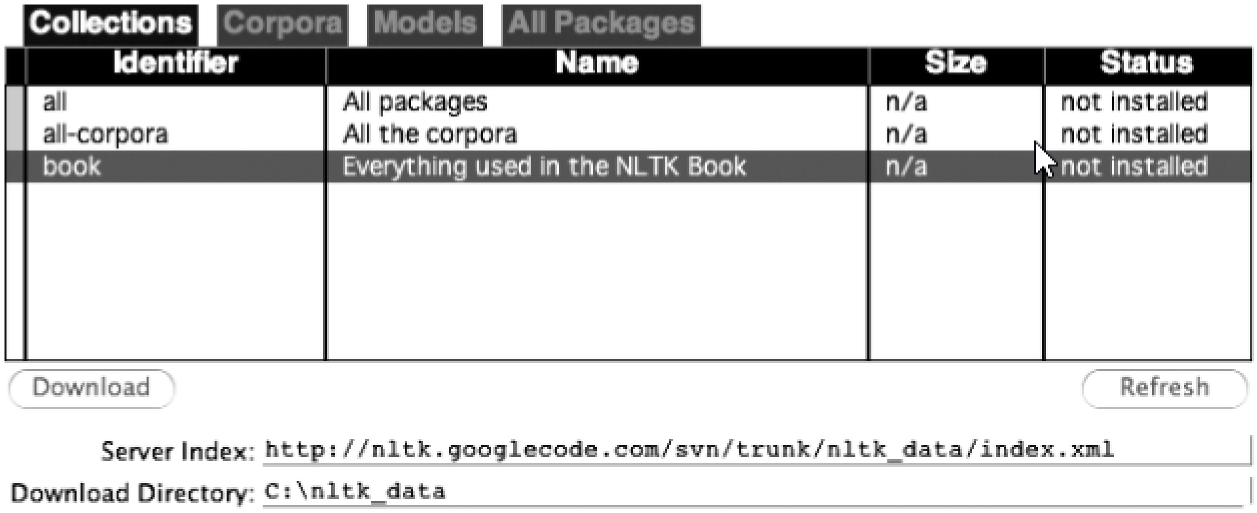
So on …..

Most of the data in our digital universe is unstructured, i.e., textual. Big data analytics techniques are normally required to analyze textual data in a production scale development of any NLP techniques, e.g., IBM Watson, Goolge search, etc.

Natural Language Toolkit (NLTK)

- Created in 2001 at the University of Pennsylvania
- It has been a widely used Python library to do basic NLP tasks
- It has now been adopted in many universities across the world
- How to use NLTK in Databricks

```
!pip install nltk
import nltk
nltk.download()
```



Identifier	Name	Size	Status
all	All packages	n/a	not installed
all-corpora	All the corpora	n/a	not installed
book	Everything used in the NLTK Book	n/a	not installed

Server Index: http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml
Download Directory: C:\nltk_data

```
>> from nltk.book import *
>> text1
<Text: Moby Dick by Herman
Melville 1851>
```

```
*** 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.
```

NLTK Modules

Language Preprocessing Task	NLTK Modules	Functionality
Accessing corpora	nltk.corpus	Standardized interfaces to textual corpora and lexicons
String processing	nltk.tokenize, nltk.stem	Tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	Statistical testing like t-test, chi-test
Part-of-speech tagging	nltk.tag	N-gram, backoff, Hidden Markov Model
Classification	nlt.classify, nltk.cluster	Decision tree, Naïve Bayes, etc.
Chunking	nltk.chunk	Regular expression, n-gram, named entity detection
Parsing	nltk.parse	Chart, probabilistic/linguistic dependency

NLTK Modules

Language Preprocessing Task	NLTK Modules	Functionality
Evaluation metrics	nltk.metrics	Precision, recall, agreement coefficients
Probability and estimation	nltk.probability	Frequency distribution, probability distribution
Applications	nltk.app, nltk.chat	Chatbots, wordnet browser, parsers
Linguistic fieldwork	nltk.toolbox	Manipulate data using third-party library like SIL framework
Semantic interpretation	nltk.sem, nltk.inference	Model checking, lambda calculus

Searching a Word in Text

- Option 1.
  - Find a word in a text. Use the “concordance” method
  - Offer insights into the surrounding contexts by showing the location/sentence where the word is found
- Option 2.
  - Find words similar to the given word. Use the “similar” method
  - Then use option 1 to analyze contexts

Searching a Word in Text

Option 1. Find a word in a text

```
import nltk
nltk.download("book")
from nltk.book import *
```

```
>> Text1
Out[5]: <Text: Moby Dick by Herman Melville 1851> // this is the entire Moby dick novel
>> text1.generate()
Building ngram index...
long , from one to the top - mast , and no coffin and went out a sea
captain -- this peaking of the whales . , so as to preserve all his
might had in former years abounding with them , they toil with their
lances , strange tales of Southern whaling . at once the bravest
Indians he was , after in vain strove to pierce the profundity . ?
then ?" a levelled flame of pale , And give no chance , watch him ;
though the line , it is to be gainsaid . have been
Out[11]: 'long , from one to the top - mast , and no coffin and went out a sea\ncaptain -- this
peaking of the whales .
>> len(text1)
Out[13]: 260819 // there are 260K words in this novel
```

Searching a Word in Text

Option 1. Find a word in a text

>> text1.concordance("monstrous")

```
Displaying 11 of 11 matches: ong the former , one was of a most monstrous size . ... This came
towards us , ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r ll
over with a heathenish array of monstrous clubs and spears . Some were thick d as you gazed ,
and wondered what monstrous cannibal and savage could ever hav that has survived the flood ;
most monstrous and most mountainous ! That Himmal they might scout at Moby Dick as a monstrous
fable , or still worse and more de th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of
Whales . I shall ere l ing Scenes . In connexion with the monstrous pictures of whales , I am
strongly ere to enter upon those still more monstrous stories of them which are to be fo ght
have been rummaged out of this monstrous cabinet there is no telling . But of Whale - Bones ;
for Whales of a monstrous size are oftentimes cast up dead u
```

A concordance function allows us to see words in context. For example, “monstrous” above appeared in different contexts like “monstrous picture” or “monstrous size”

Option 2. Find similar words

>> text1.similar("monstrous")

```
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless
```

>> text1.similar("monstrous")

```
very so exceedingly heartily a as good great extremely remarkably sweet vast amazingly
```

Let’s explore a bit while the words similar to “monstrous” are different in the two texsts

Searching a Word in Text

Option 2. Find words similar to a word within a text

>> text2

```
Out[16]: <Text: Sense and Sensibility by Jane Austen 1811>
```

>> text2.generate()

```
Building ngram index... knew , had by this remembrance , and if , by rapid degrees , so long . ,
she could live without one another , and in her rambles . at least the last evening of a brother
, could you know , from the first . Dashwood ? this gentleman himself , and must put up with a
kindness which they are very much vexed at , for it -- …'
```

>> len(text2)

```
Out[19]: 141576
```

>> text2.concordance(“monstrous”)

```
Displaying 11 of 11 matches: . " Now , Palmer , you shall see a monstrous pretty girl ." He
immediately went your sister is to marry him . I am monstrous glad of it , for then I shall have
ou may tell your sister . She is a monstrous lucky girl to get him , upon my ho k how you will
like them . Lucy is monstrous pretty , and so good humoured and Jennings , " I am sure I shall
be monstrous glad of Miss Marianne ' s company usual noisy cheerfulness , " …
```

The word “monstrous” is used differently in the two novels. Austen uses this word within pride or social contexts and it has mostly a negative connotation, while Melville (Moby dick) uses the word to discuss the whale size or other aspects and it has a mostly positive connotation. This means, “similar” method in nltk shows word that appear mostly together within a range of contexts for a given a word.

Searching a Word in Text

“common_contexts” method can be used to see how multiple words are used together in a given text

```
>> text1.common_contexts(["monstrous", "very"])
No common contexts were found
>> text2.common_contexts(["monstrous", "very"])
am_glad a_pretty a_lucky is_pretty be_glad

>> text1.common_contexts(["monstrous", "fearless"])
most_and
>> text2.common_contexts(["monstrous", "fearless"])
('The following word(s) were not found:', 'monstrous fearless')
```

Exploring Simple Statistics about Lexicons/Words in a Text

```
def get_length(text):
  len_all = len(text)
  len_unique = len(set(text))
  tt = [t.lower() for t in text]
  len_unique_lower = len(set(tt))
  print("Name = %s. Total Length = %d. Total unique = %d.
         Total unique lower = %d"%(text.name, len_all, len_unique, len_unique_lower))
  return len_all, len_unique, len_unique_lower
text1_len_all, text1_len_unique, len_unique_lower = get_length(text1)
text2_len_all, text2_len_unique, len_unique_lower = get_length(text2)
text3_len_all, text3_len_unique, len_unique_lower = get_length(text3)
```

```
Name = Moby Dick by Herman Melville 1851. Total Length = 260819. Total unique = 19317. Total
unique lower = 17231 Name = Sense and Sensibility by Jane Austen 1811. Total Length = 141576.
Total unique = 6833. Total unique lower = 6403 Name = The Book of Genesis. Total Length = 44764.
Total unique = 2789. Total unique lower = 2628
```

Exploring Simple Statistics about Lexicons/Words in a Text

```
# how diverse is the usage of different words in a given text?
from __future__ import division
def lexical_diversity_of_entire_text(text):
  len_all = len(text)
  len_unique = len(set(text))
  diversity = len_unique*100.0/len_all
  print ("Name = %s. Lexical Diversity = %.2f%%"%(text.name, diversity))
  #return diversity
lexical_diversity_of_entire_text(text1)
lexical_diversity_of_entire_text(text2)
lexical_diversity_of_entire_text(text3)
```

```
Name = Moby Dick by Herman Melville 1851. Lexical Diversity = 7.41%
Name = Sense and Sensibility by Jane Austen 1811. Lexical Diversity = 4.83%
Name = The Book of Genesis. Lexical Diversity = 6.23%
```

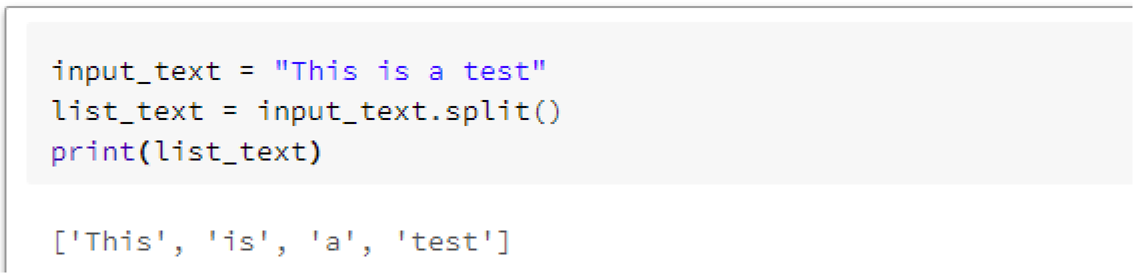
Exploring Simple Statistics about Lexicons/Words in a Text

```
# how frequently does a word appear in a text compared to the rest of the words in the text?
def compute_word_percentage_in_text(text, word):
  word_count = text.count(word)
  text_len = len(text)
  word_pct = word_count*100.0/text_len
  print ("Name = %s. Word = %s. Percentage = %.2f%%"%(text.name, word, word_pct))
compute_word_percentage_in_text(text1, "a")
compute_word_percentage_in_text(text2, "a")
compute_word_percentage_in_text(text3, "a")
```

```
Name = Moby Dick by Herman Melville 1851. Word = a. Percentage = 1.75%
Name = Sense and Sensibility by Jane Austen 1811. Word = a. Percentage = 1.44%
Name = The Book of Genesis. Word = a. Percentage = 0.76%
```

Creating Data Structures from Input Text

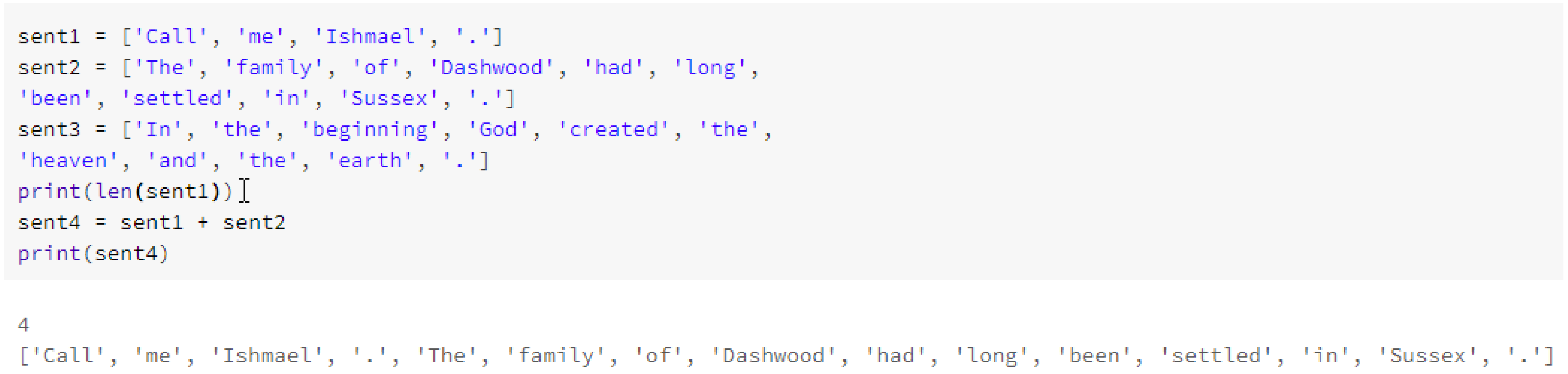
If we create a Python list from an input text, we can do many any Python operation with the list



```
input_text = "This is a test"
list_text = input_text.split()
print(list_text)
```

```
['This', 'is', 'a', 'test']
```

Two sentences as lists can be combined.



```
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', '.']
print(len(sent1))
sent4 = sent1 + sent2
print(sent4)
```

```
4
['Call', 'me', 'Ishmael', '.', 'The', 'family', 'of', 'Dashwood', 'had', 'long', 'been', 'settled', 'in', 'Sussex', '.']
```

NLTK Basic Functions

Creating Data Structures from Input Text

A sentence can be appended with more words

```
sent1 = ['Call', 'me', 'Ishmael', '.']
sent1.append("no call me Abraham".split())
sent1
```

```
Out[65]: ['Call', 'me', 'Ishmael', '.', ['no', 'call', 'me', 'Abraham']]
```

We can check the position of word in a sentence that is converted to a list

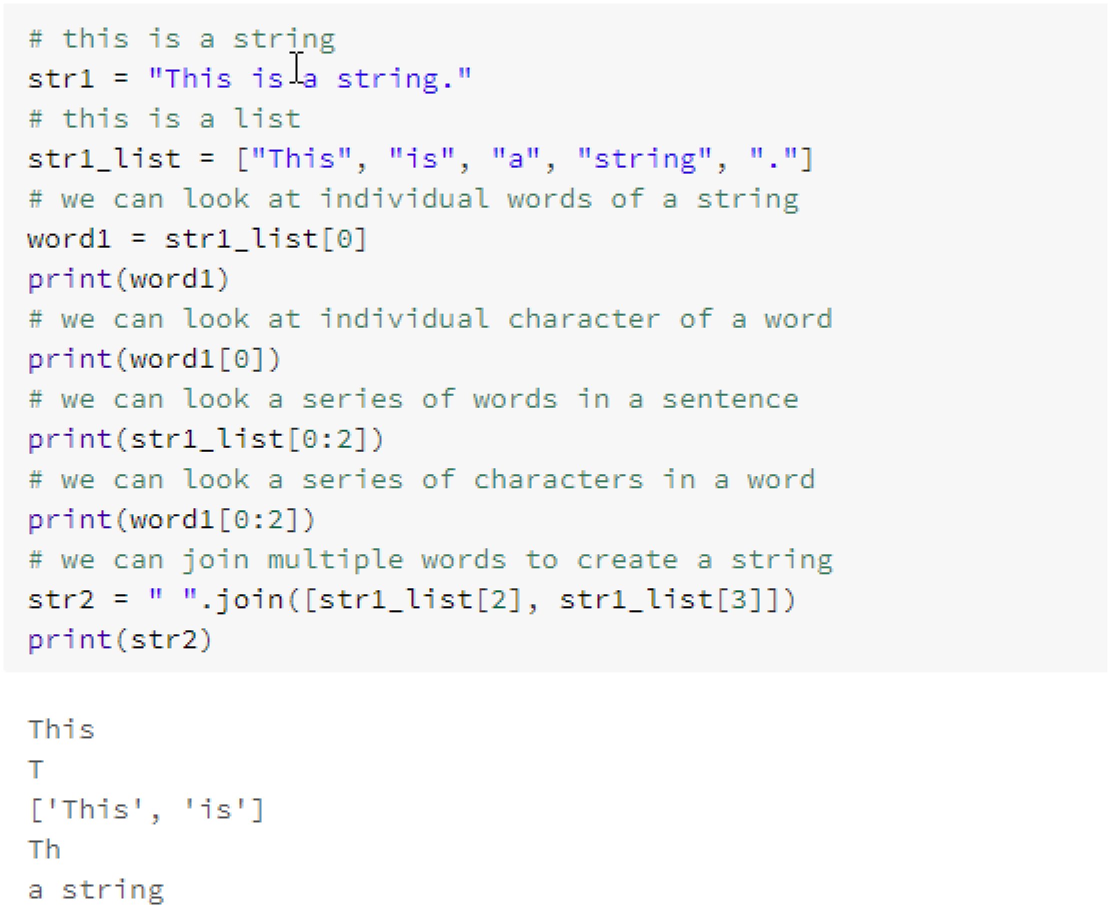
```
sent1 = ['Call', 'me', 'Ishmael', '.']
sent1 = [a.strip().lower() for a in sent1]
words_to_append = "no call me Abraham".split()
for w in words_to_append:
  sent1.append(w)
```

```
def get_word_stat_in_text(text, word):
  total_freq = text.count(word)
  first_occurrence_pos = text.index(word)
  print("Total Freq = %d. First Position = %d"%(total_freq, first_occurrence_pos))
```

```
get_word_stat_in_text(sent1, "call")
```

```
Total Freq = 2. First Position = 0
```

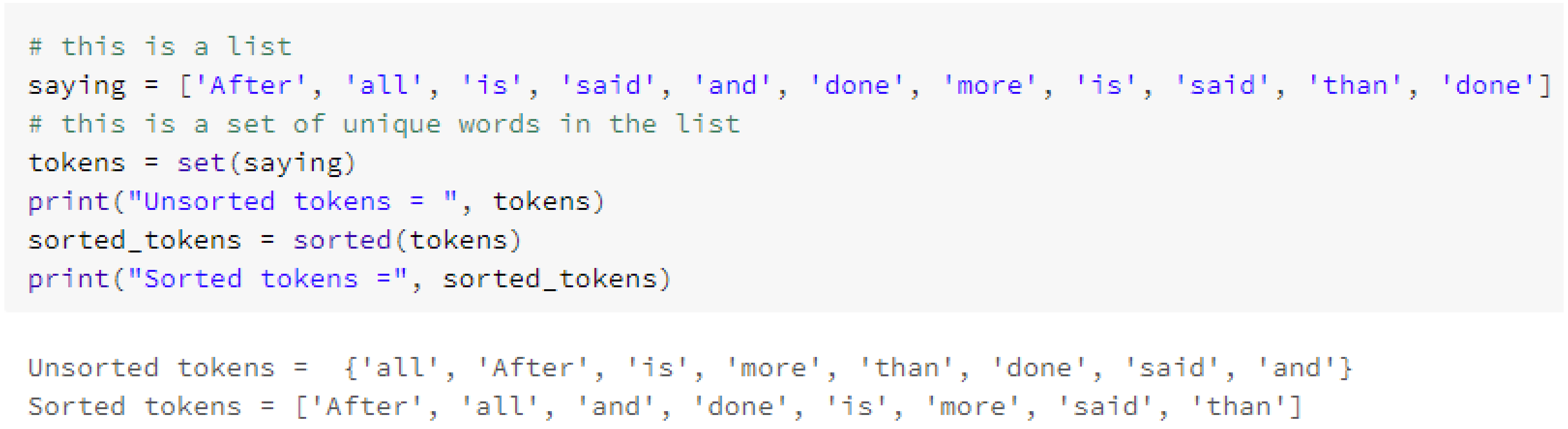
Variables and Basic Operations



```
# this is a string
str1 = "This is a string."
# this is a list
str1_list = ["This", "is", "a", "string", "."]
# we can look at individual words of a string
word1 = str1_list[0]
print(word1)
# we can look at individual character of a word
print(word1[0])
# we can look a series of words in a sentence
print(str1_list[0:2])
# we can look a series of characters in a word
print(word1[0:2])
# we can join multiple words to create a string
str2 = " ".join([str1_list[2], str1_list[3]])
print(str2)
```

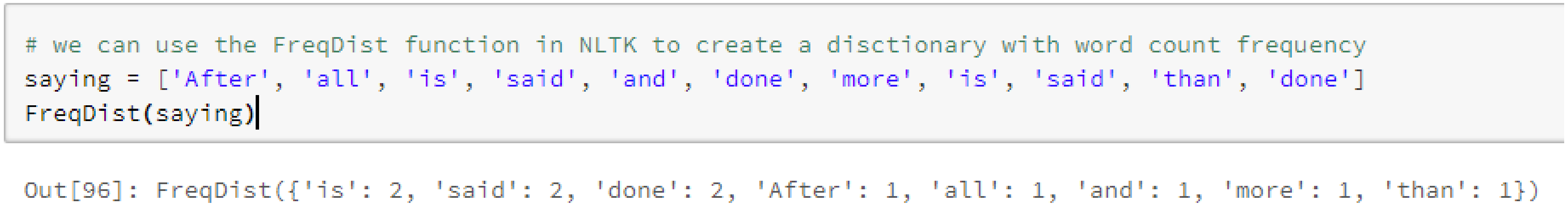
```
This
T
['This', 'is']
Th
a string
```

Variables and Basic Operations



```
# this is a list
saying = ['After', 'all', 'is', 'said', 'and', 'done', 'more', 'is', 'said', 'than', 'done']
# this is a set of unique words in the list
tokens = set(saying)
print("Unsorted tokens = ", tokens)
sorted_tokens = sorted(tokens)
print("Sorted tokens =", sorted_tokens)
```

```
Unsorted tokens =  {'all', 'After', 'is', 'more', 'than', 'done', 'said', 'and'}
Sorted tokens = ['After', 'all', 'and', 'done', 'is', 'more', 'said', 'than']
```



```
# we can use the FreqDist function in NLTK to create a disctionary with word count frequency
saying = ['After', 'all', 'is', 'said', 'and', 'done', 'more', 'is', 'said', 'than', 'done']
FreqDist(saying)
```

```
Out[96]: FreqDist({'is': 2, 'said': 2, 'done': 2, 'After': 1, 'all': 1, 'and': 1, 'more': 1, 'than': 1})
```

Word Collocations

A collocation is a sequence of words that occur together often. For example, “red wine” is a collocation of two words “red” and “wine”. A collocation of two words is called a bigram.

Books and texts available via nltk corpora have “collocations” method to check for bigrams.

```
text1.collocations()
```

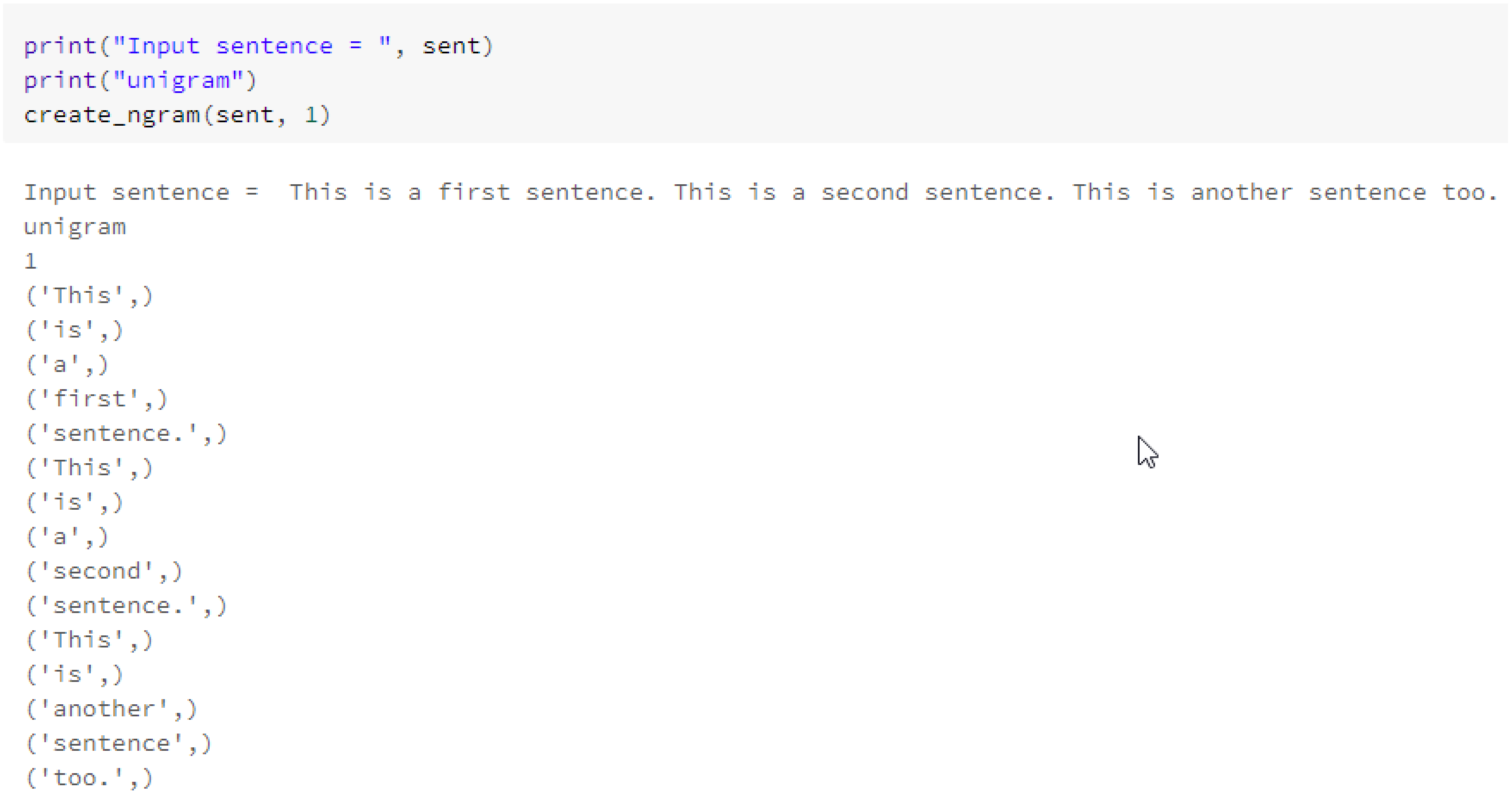
```
Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab; sperm
whale; Right Whale; Captain Peleg; New Bedford; Cape Horn; cried Ahab;
years ago; lower jaw; never mind; Father Mapple; cried Stubb; chief
mate; white whale; ivory leg; one hand
```

We can use the nltk ngrams method to create different ngrams. A list of individual words is called a unigram, two consecutive words is called a bigram, three consecutive words is called a trigram. Let’s see how it works!

```
from nltk.util import ngrams
sent = 'This is a first sentence. This is a second sentence. This is another sentence too.'
def create_ngram(sentence, n):
  ngs = ngrams(sentence.split(), n)
  print(n)
  for item in ngs:
    print(item),
```

NLTK Basic Functions

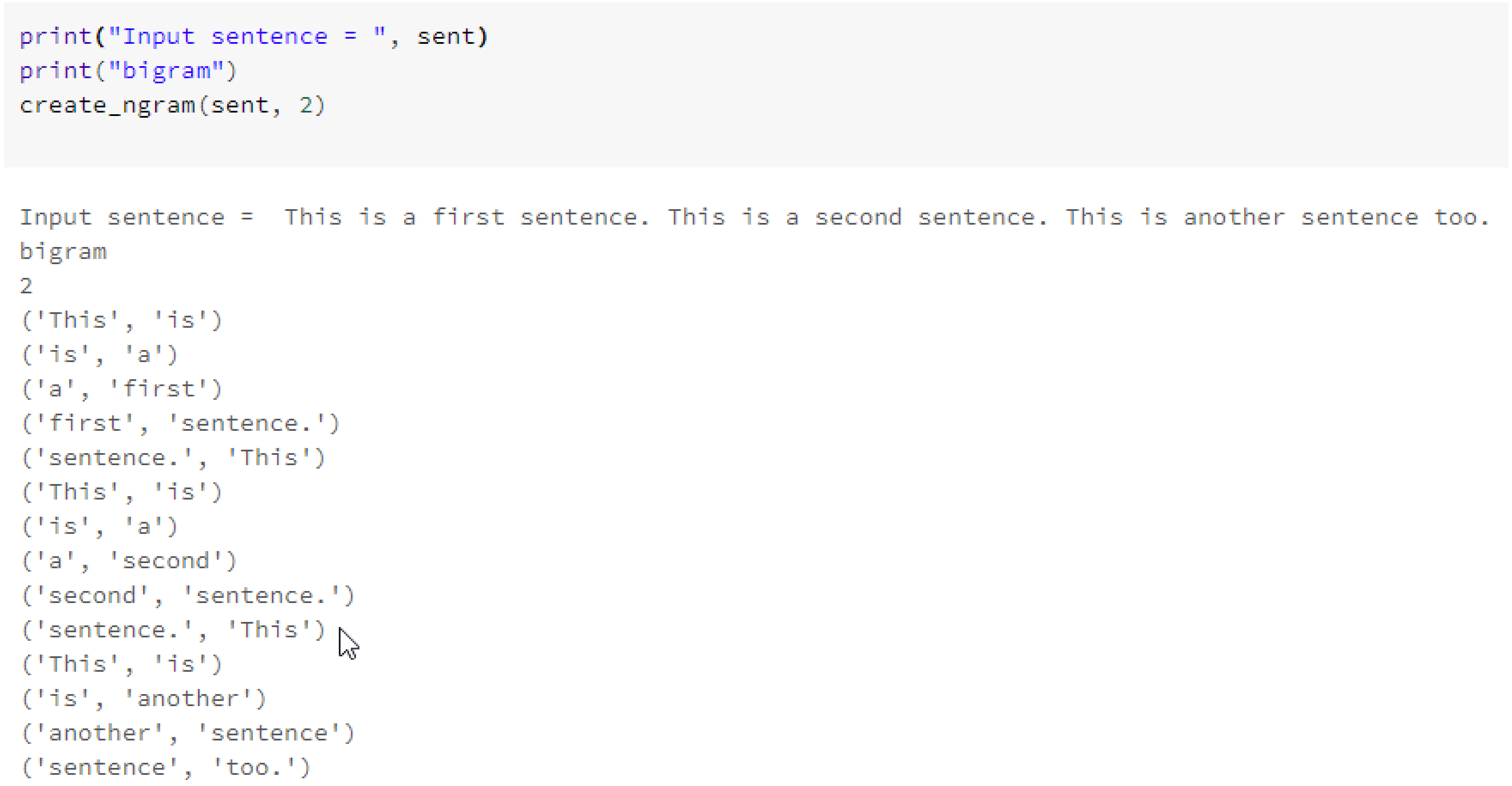
Word Collocations



```
print("Input sentence = ", sent)
print("unigram")
create_ngram(sent, 1)
```

```
Input sentence =  This is a first sentence. This is a second sentence. This is another sentence too.
unigram
1
('This',)
('is',)
('a',)
('first',)
('sentence.',)
('This',)
('is',)
('a',)
('second',)
('sentence.',)
('This',)
('is',)
('another',)
('sentence',)
('too.',)
```

Word Collocations



```
print("Input sentence = ", sent)
print("bigram")
create_ngram(sent, 2)
```

```
Input sentence =  This is a first sentence. This is a second sentence. This is another sentence too.
bigram
2
('This', 'is')
('is', 'a')
('a', 'first')
('first', 'sentence.')
('sentence.', 'This')
('This', 'is')
('is', 'a')
('a', 'second')
('second', 'sentence.')
('sentence.', 'This')
('This', 'is')
('is', 'another')
('another', 'sentence')
('sentence', 'too.')
```

Word Collocations

```
print("Input sentence = ", sent)
print("trigram")
create_ngram(sent, 3)
```

```
Input sentence =  This is a first sentence. This is a second sentence. This is another sentence too.
trigram
3
('This', 'is', 'a')
('is', 'a', 'first')
('a', 'first', 'sentence.')
('first', 'sentence.', 'This')
('sentence.', 'This', 'is')
('This', 'is', 'a')
('is', 'a', 'second')
('a', 'second', 'sentence.')
('second', 'sentence.', 'This')
('sentence.', 'This', 'is')
('This', 'is', 'another')
('is', 'another', 'sentence')
('another', 'sentence', 'too.')
```

How to use NLTK or Custom Python Functions within PySpark

Option 1. use Pyspark + UDF. UDF = User defined function

Step 1. Write your function with an annotation @udf as follows

```
from pyspark.sql.functions import udf

@udf
def preprocess_text(text):
        // do your text processing say using nltk
        return processed_text
```

Step 2. say your texts are loaded into a pyspark dataframe df, where you have a column “text” whose contents you want to preprocess using the preprocess_text method

```
dfText = df.select(“text”, preprocess_text(“text”).alias(“text_preprocessed”)
```

How to use NLTK or Custom Python Functions within PySpark

Option 2. use lambda within an RDD

Step 1. Create a list of items from your input data (e.g., from a sentence, create a list of words). Then create and RDD of the list

```
words = ["This", "is", "A", "Test"]
rdd = sc.parallelize(words)
```

Step 2. Write your one custom python function (e.g., make lowercase of each word)

```
def toLower(word):
   return word.lower()
```

Step 3. Now call the map function from the rdd on each word

```
rdd2 = rdd.map(lambda w: toLower(w))
```

Reference

Part of the lecture slides are an adaptation of Section 1 from the following book

“Steven Bird, Ewen Klein, and Edward Loper. Natural Language Processing with Python. O’Reilly, 2021”