

# DATA SOCIETY®

The premiere data science training for professionals

*“If you can’t explain it simply, you don’t understand it well enough.”*

- Albert Einstein

# Which problems will you solve?

---

Objectives for this lecture:

1. Automatically extract key words and phrases that sum up the style and content of a text
2. Tabulate word frequencies and distributions
3. Plot and compare metrics across texts

# Overview

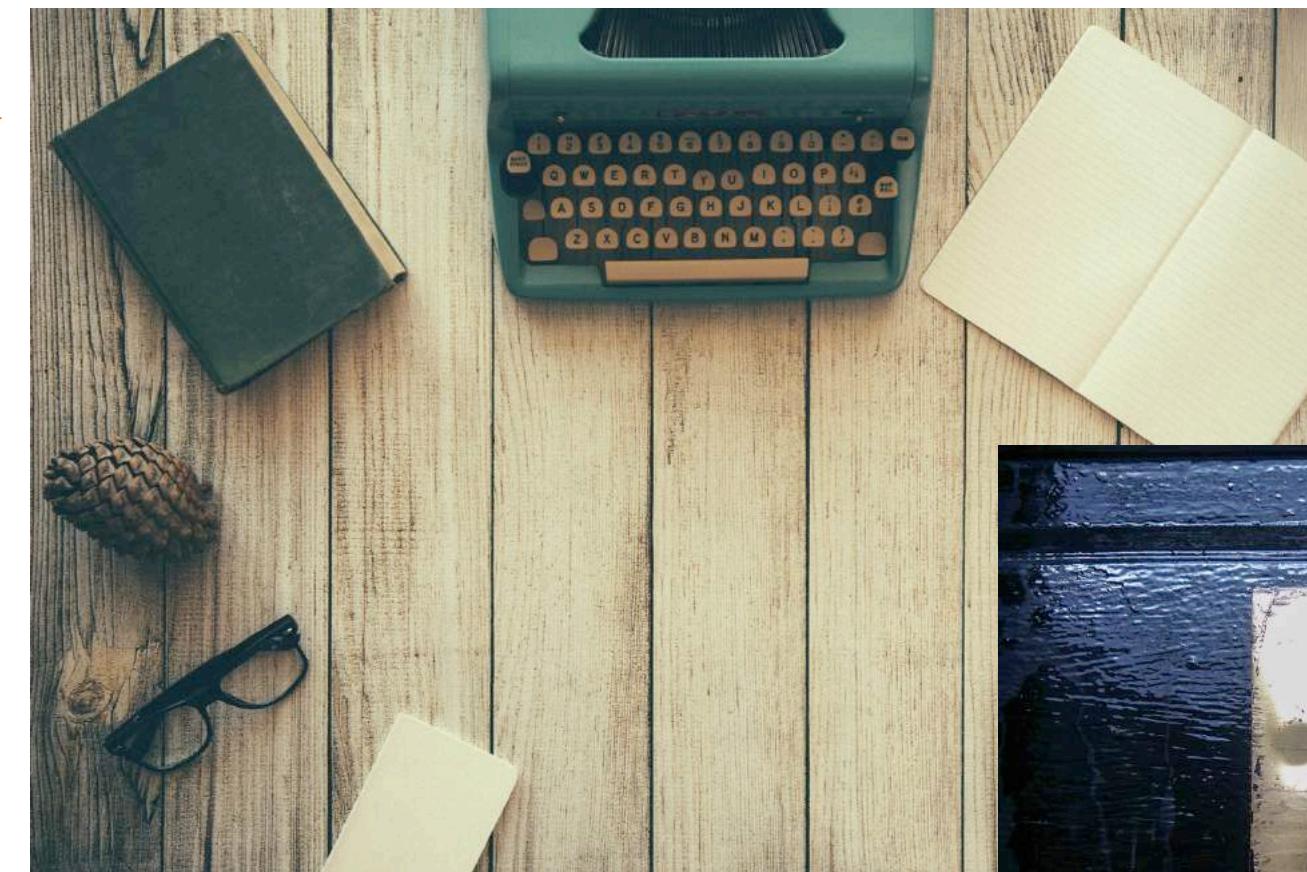
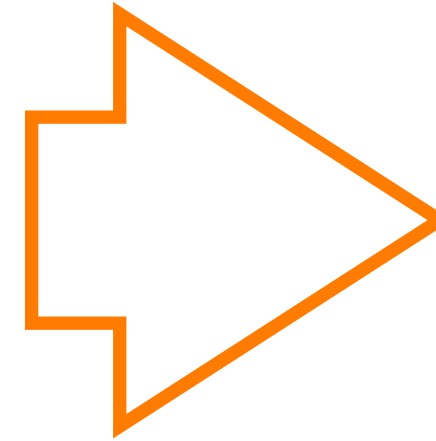
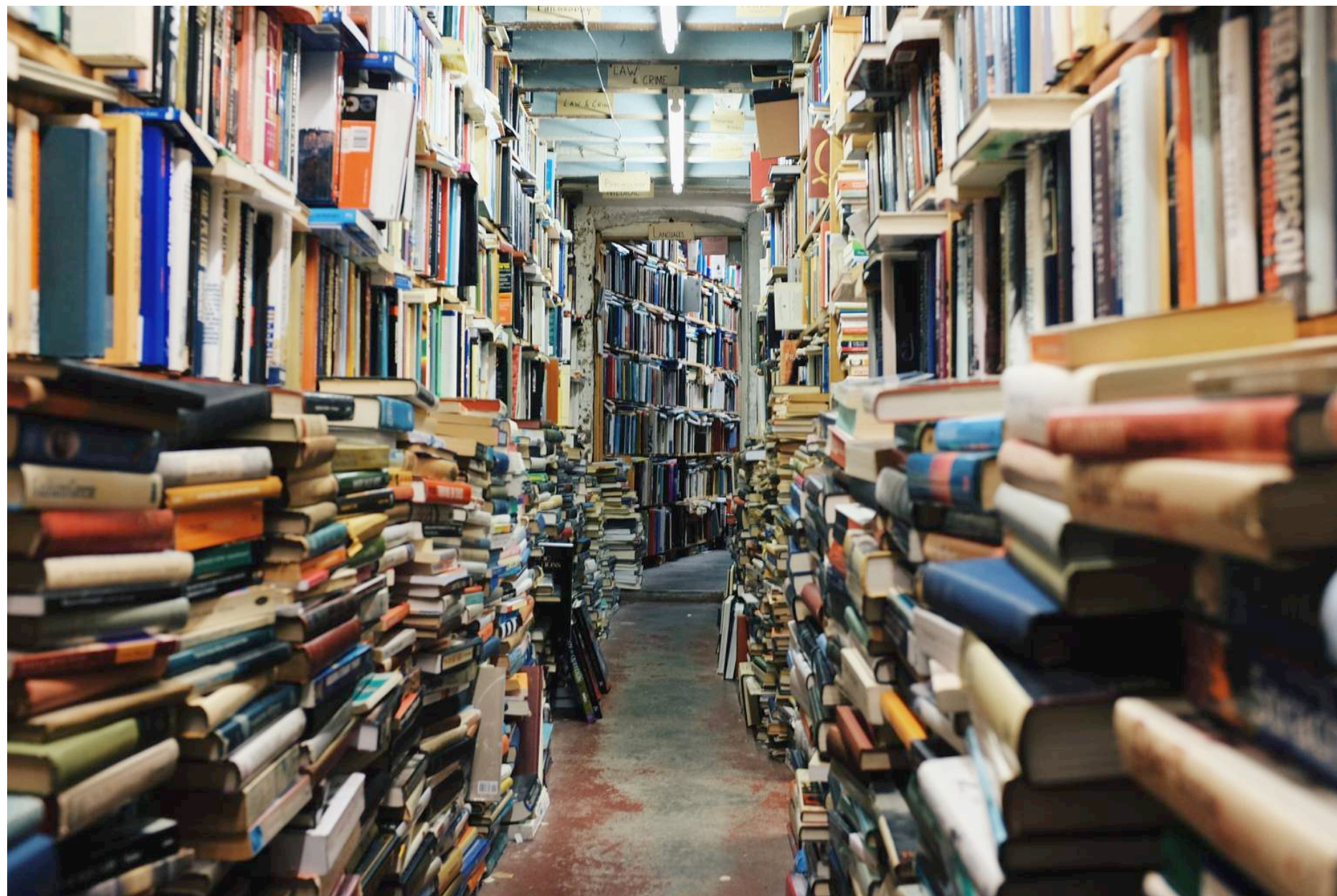
---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions



# Why text mining?

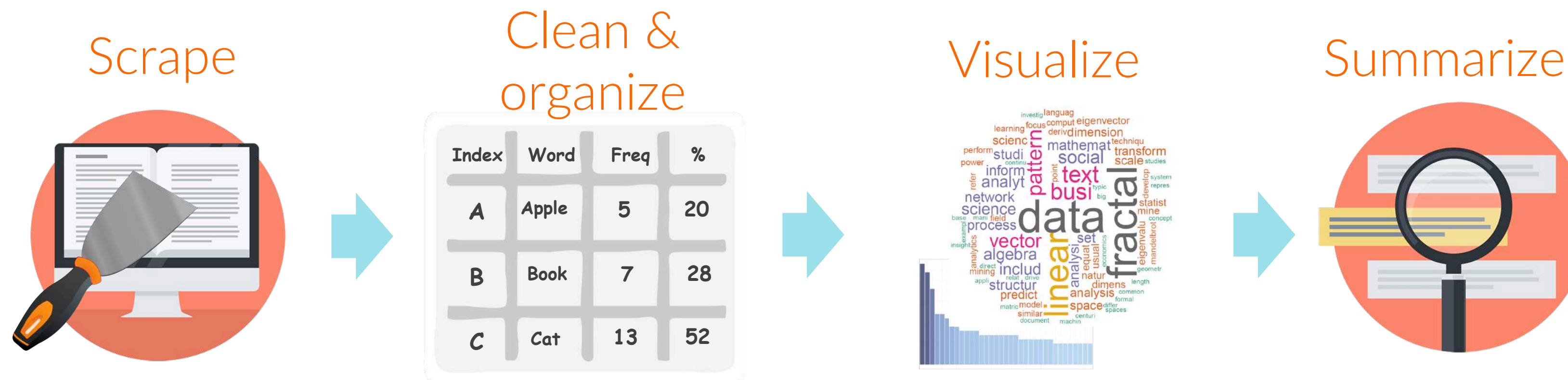
Most of the information and knowledge in the world is stored as text





# Text mining order of events

## Intro to Text Mining



# Additional text mining capabilities

---

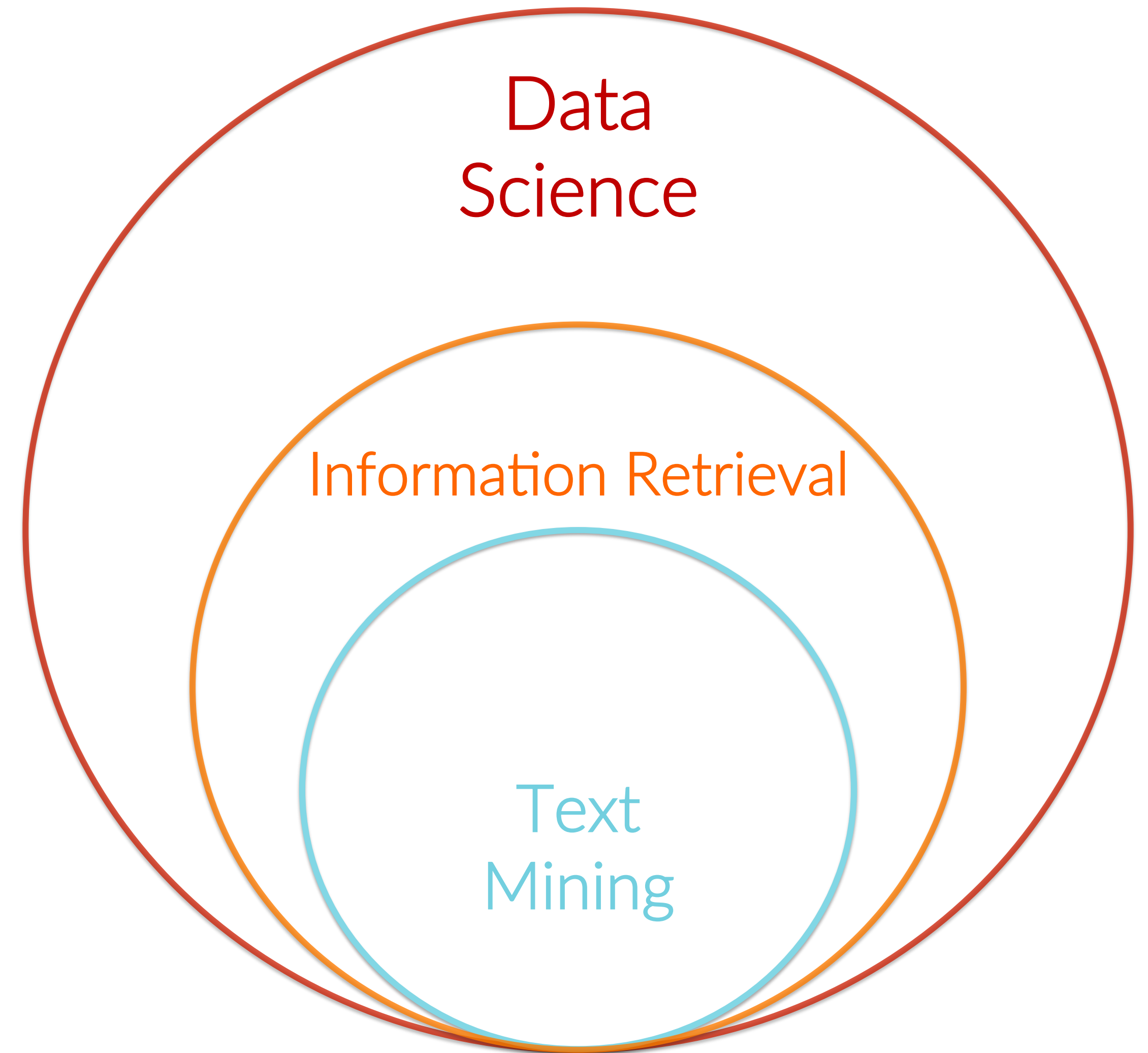
Other modules...



# What is text mining?

---

- Process of getting insightful and valuable information out of text data
- Part of Information Retrieval
- Information Retrieval is part of Data Science





# What is text mining?

---

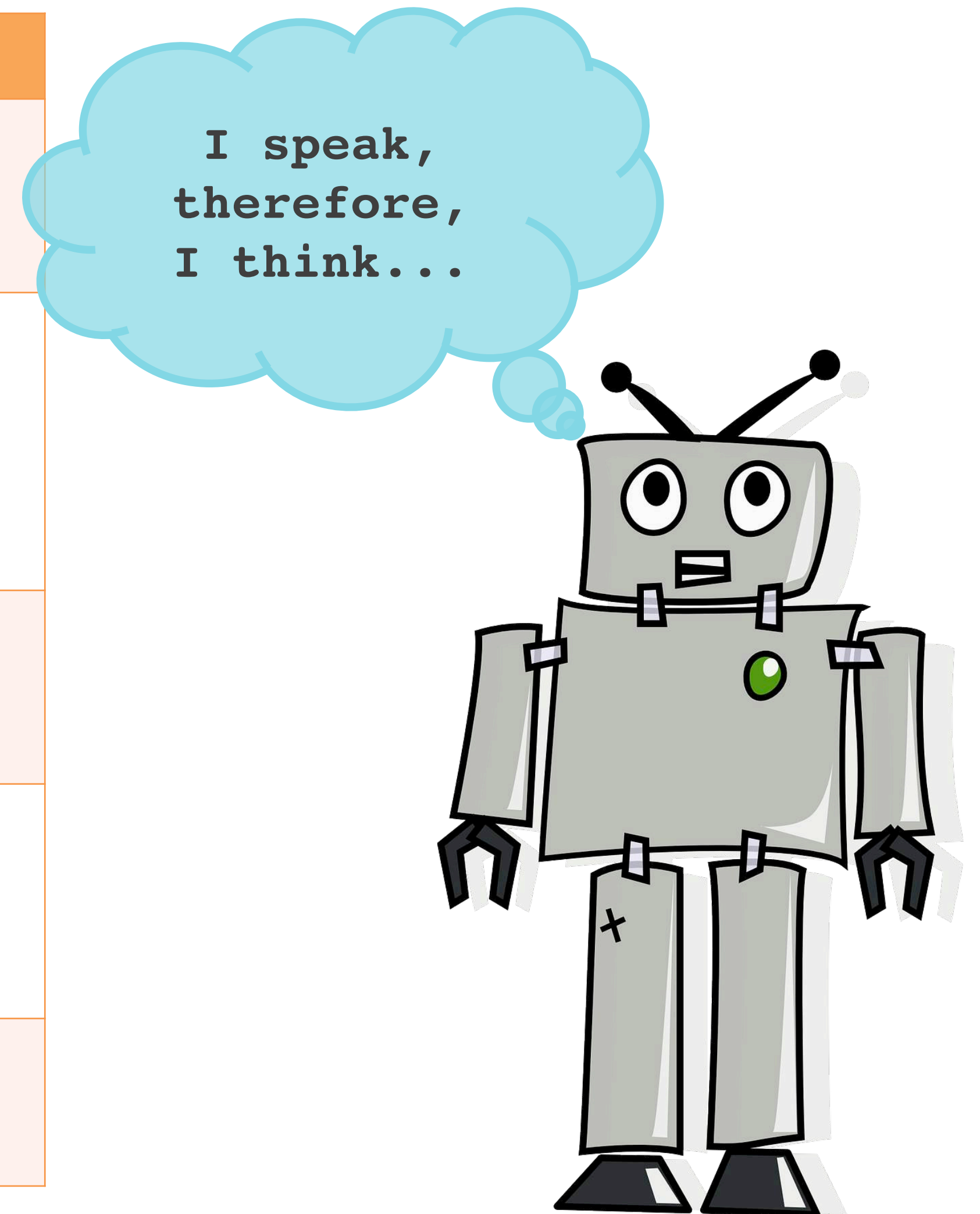
Text mining employs methods from various fields including mathematics, statistics, computational linguistics, and programming.





# Text mining: some use cases

Field	Text Mining
Natural language processing	<ul style="list-style-type: none"><li>• Discover patterns in speech</li><li>• Understand how human language works</li></ul>
Business	<ul style="list-style-type: none"><li>• Help increase profits by analyzing product reviews</li><li>• Summarize and group reports, catalogue documents</li></ul>
Psychology & psychiatry	<ul style="list-style-type: none"><li>• Analyze sentiments to detect and prevent dangerous social behavior</li></ul>
Network analysis	<ul style="list-style-type: none"><li>• Augment network analysis to better understand how people interact within a group or react to a certain event</li></ul>
Many more ...	...



# Text in text mining

Letters

Words

Sentences

Documents

d  
r  
w  
o



sentence  
word is  
This a not



To make a whole document.

This word is not a sentence.

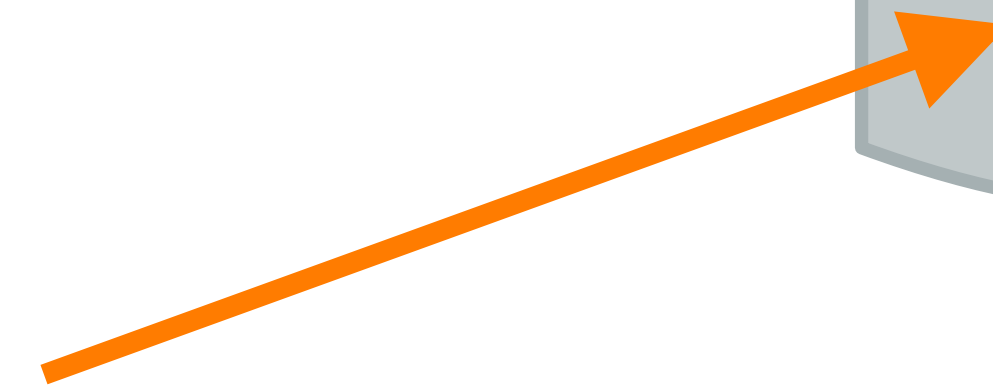
I will type it anyway.



Nonsensical Document

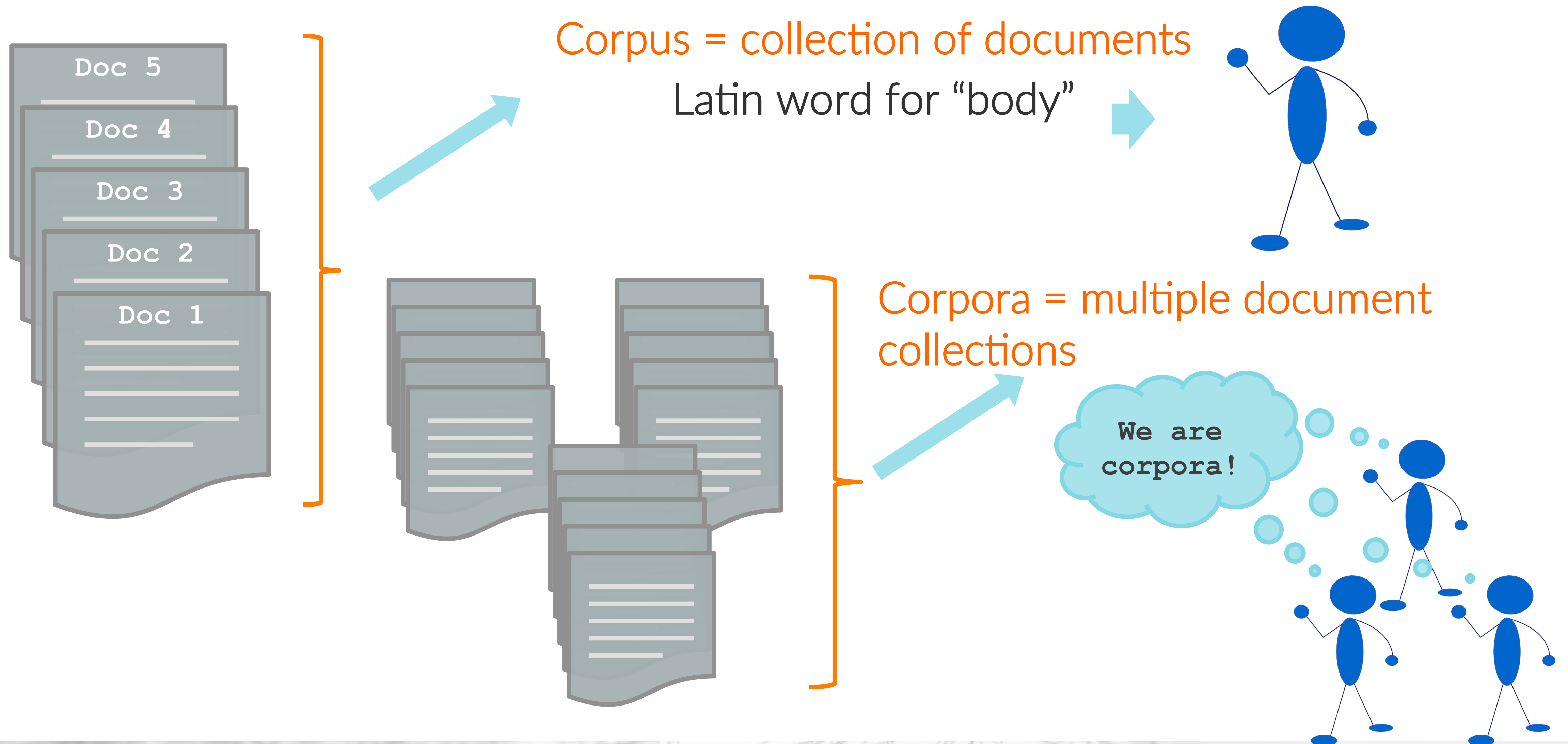
This word is not a sentence. I will type it anyway. To make a whole document.

Text = Document ?





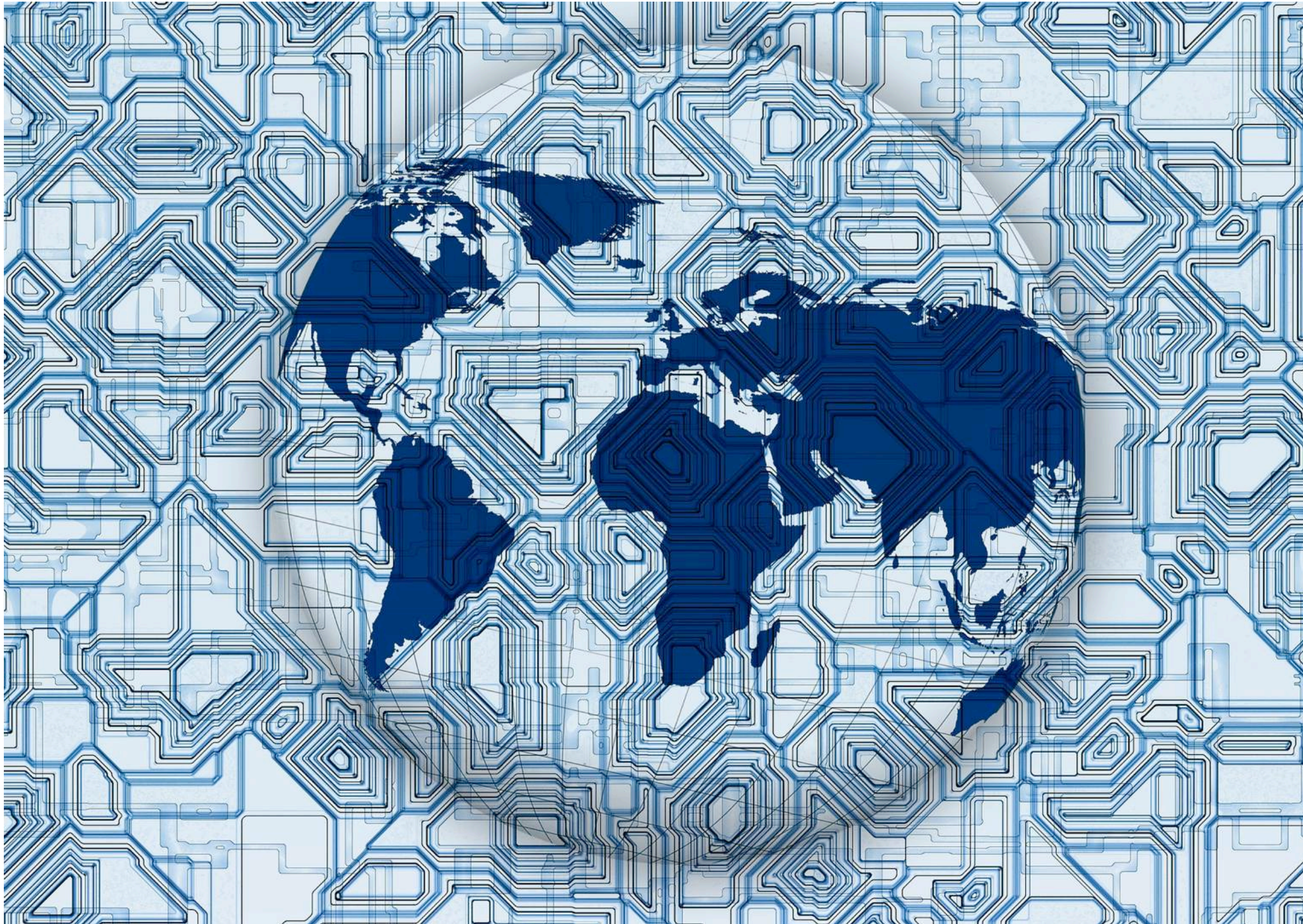
# Text in text mining





# How to get text data?

---



Web scraping - retrieving data from an online source, usually a web page, is often associated with text mining.



# Overview

---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions

# What is NLTK?

---

- NLTK is the Natural Language Toolkit, built in 2001 as part of a computational linguistics course at University of Pennsylvania
- Built with four goals in mind:
  1. **Simplicity:** to provide an intuitive framework along with building blocks to give users a practical knowledge of NLP without getting bogged down in the tedious house-keeping
  2. **Consistency:** to provide a uniform framework with consistent interfaces and data structures, and easily-guessable method names
  3. **Extensibility:** to provide a structure into which new software modules can be easily accommodated
  4. **Modularity:** to provide components that can be used independently without needing to understand the rest of the toolkit



# Some tasks that NLTK does

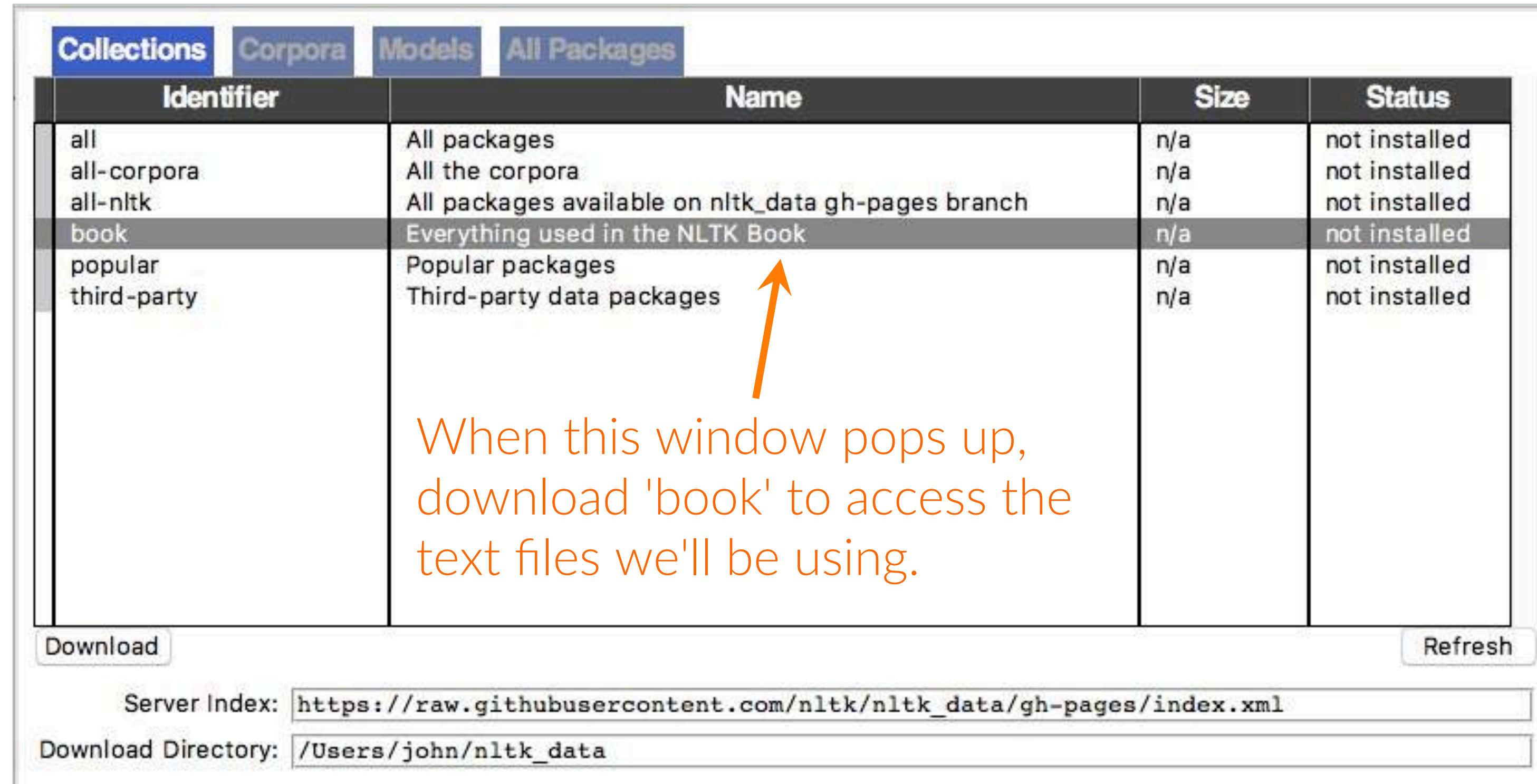
---

Language processing task	NLTK module	Functionality
Accessing corpora	corpus	standardized interfaces to corpora and lexicons
String processing	tokenize, stem	tokenizers, sentence tokenizers, stemmers
Part-of-speech tagging	tag	n-gram, backoff, Brill, HMM, TnT
Machine learning	classify, cluster, tbl	decision tree, maximum entropy, naive Bayes, EM, k-means
Parsing	parse, ccg	chart, feature-based, unification, probabilistic, dependency
Probability and estimation	probability	frequency distributions, smoothed probability distributions

# Importing nltk

```
# Import nltk as follows:  
import nltk  
nltk.download()
```

Script



The screenshot shows the NLTK Collections window with the 'All Packages' tab selected. The table lists various collections, with 'book' highlighted. An orange arrow points to the 'book' row, and a text box explains that this is the collection to download for accessing the text files used in the course.

Identifier	Name	Size	Status
all	All packages	n/a	not installed
all-corpora	All the corpora	n/a	not installed
all-nltk	All packages available on nltk_data gh-pages branch	n/a	not installed
book	Everything used in the NLTK Book	n/a	not installed
popular	Popular packages	n/a	not installed
third-party	Third-party data packages	n/a	not installed

When this window pops up, download 'book' to access the text files we'll be using.

Download Refresh

Server Index:

Download Directory:



# Importing text data

```
# Import text excerpts to use for this session:
```

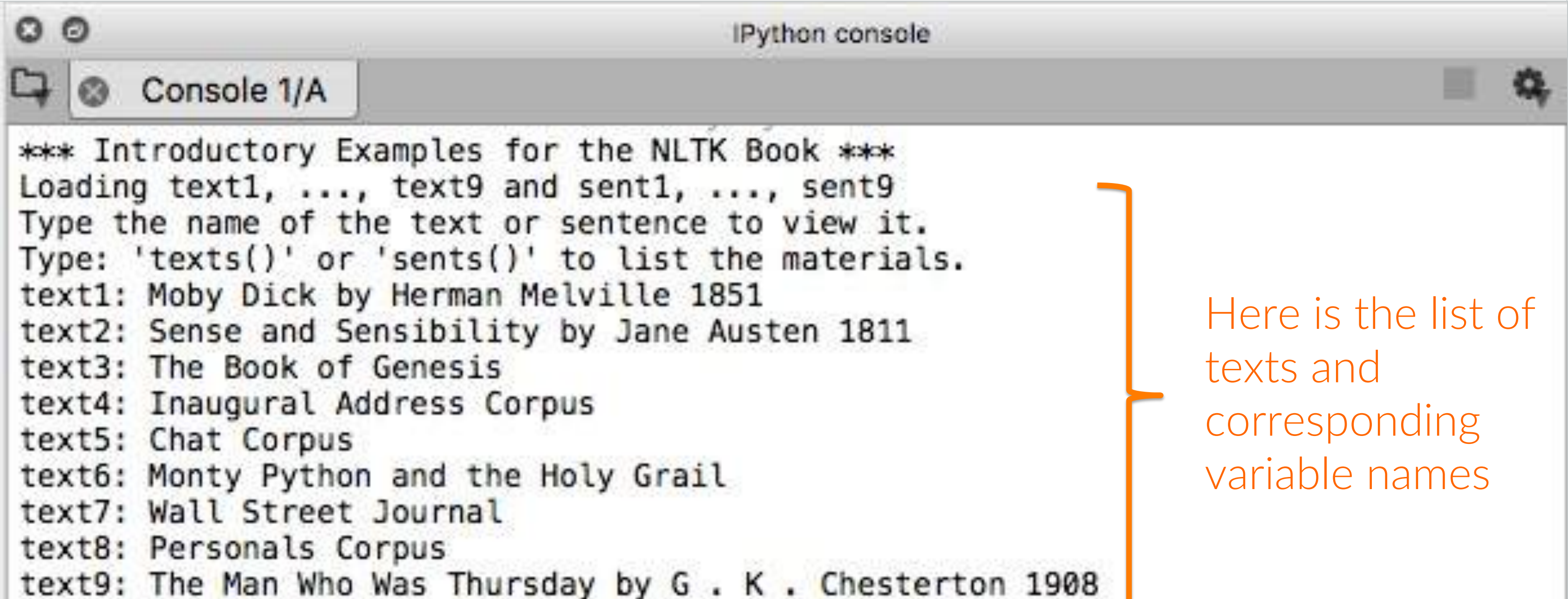
```
from nltk.book import *
```

```
# You can pull out specific texts by referencing the variable name  
text1
```

```
# <Text: Moby Dick by Herman Melville 1851>
```

```
# You can use texts()  
# to look up all the  
# text names  
texts()
```

Script



```
*** 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
```

Here is the list of  
texts and  
corresponding  
variable names



# Searching text

- A **concordance** view gives us all the occurrences of a particular word surrounded by its context.

```
# Let's pull the word 'monstrous' from the Moby Dick text.
```

Script

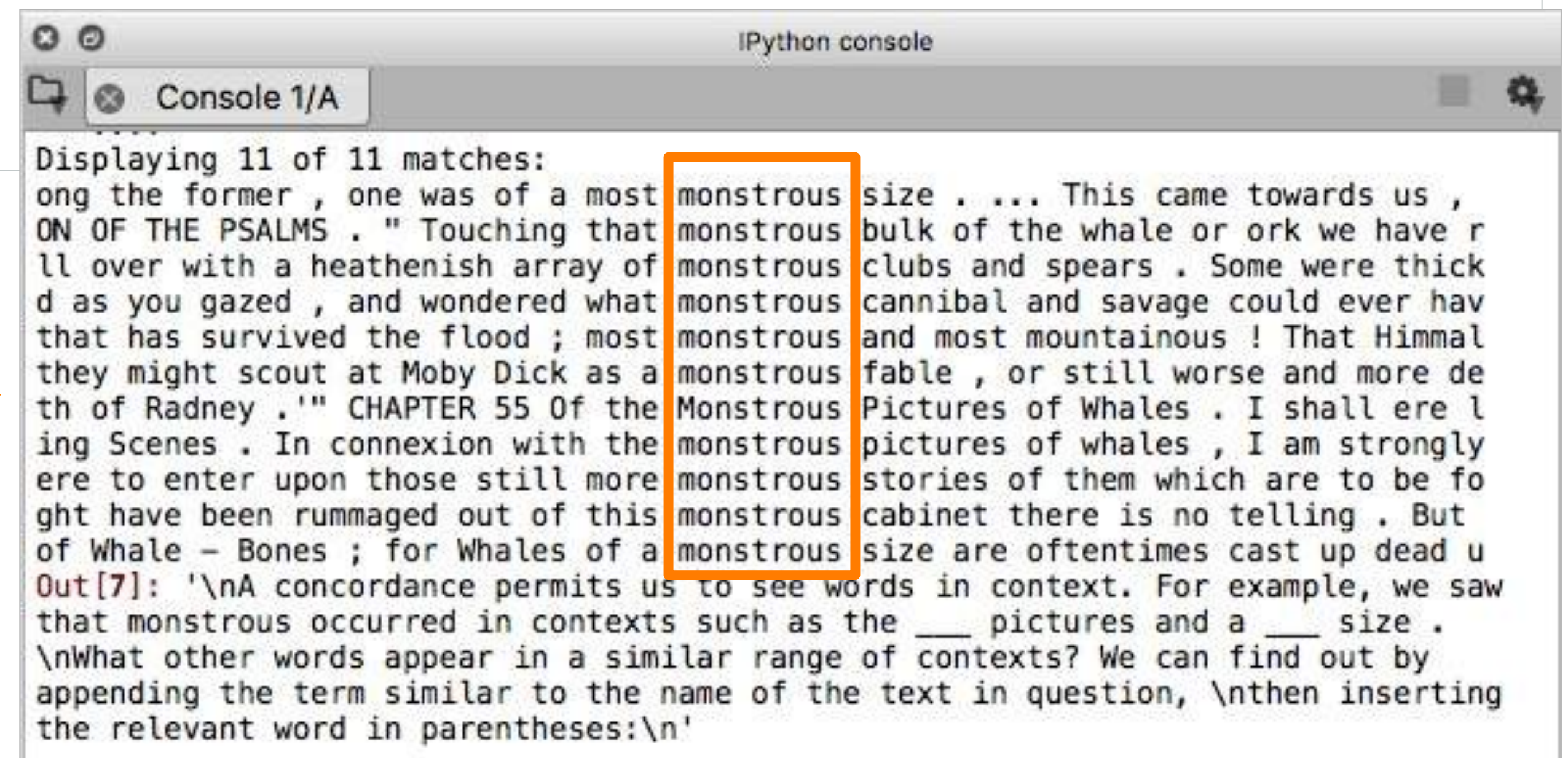
Text we  
want to use

The view  
we want

The word to  
pull out

↓ ↓ ↓  
`text1.concordance("monstrous")`

Here is the word we  
specified with the words  
that appear before and  
after each occurrence



```
IPython console
Console 1/A
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
Out[7]: '\nA concordance permits us to see words in context. For example, we saw
that monstrous occurred in contexts such as the __ pictures and a __ size .
\nWhat other words appear in a similar range of contexts? We can find out by
appending the term similar to the name of the text in question, \nthen inserting
the relevant word in parentheses:\n'
```



# Finding similar words in context

- A **similar** view gives us additional words that appear in similar locations to the word we specify

```
# Let's find words that appear similarly to "monstrous" in "Moby Dick".
```

Script

Text we  
want to use

The view  
we want

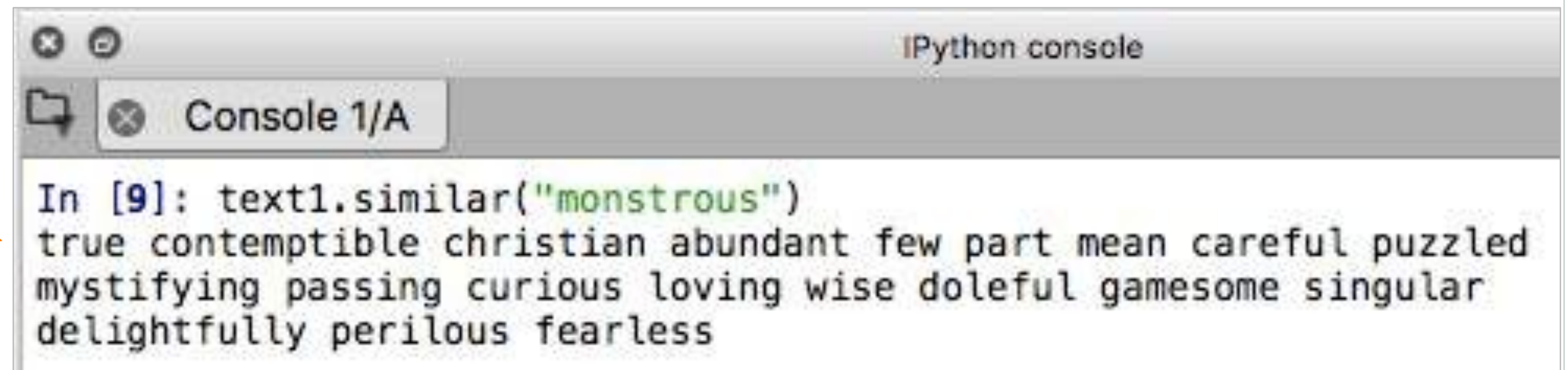
The word to reference for similar  
words in context

text1.similar("monstrous")

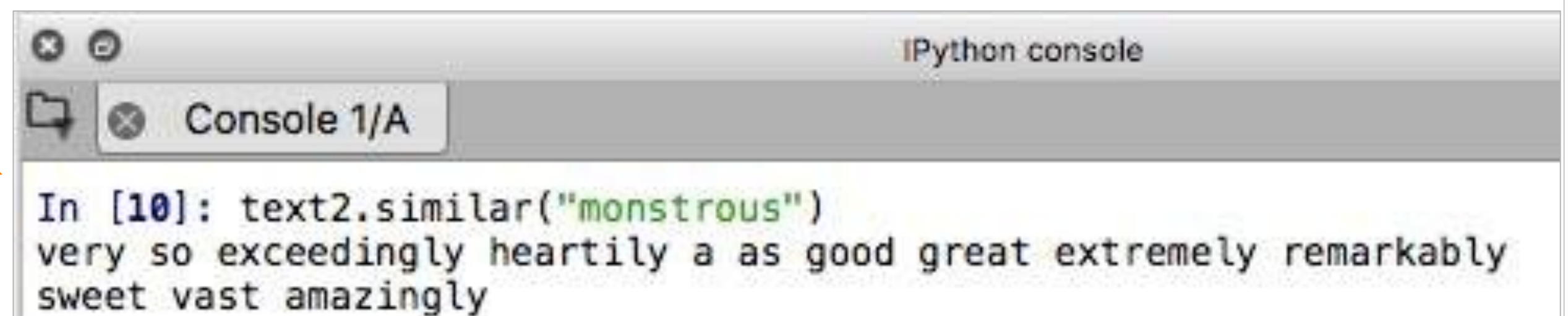
```
# What words appear near "monstrous"  
# in Jane Austen's text?
```

text2.similar("monstrous")

It looks like these words are used  
in different ways in these texts!



```
Python console  
Console 1/A  
In [9]: text1.similar("monstrous")  
true contemptible christian abundant few part mean careful puzzled  
mystifying passing curious loving wise doleful gamesome singular  
delightfully perilous fearless
```



```
Python console  
Console 1/A  
In [10]: text2.similar("monstrous")  
very so exceedingly heartily a as good great extremely remarkably  
sweet vast amazingly
```

# Finding shared context

- `common_contexts` allows us to examine just the contexts that are shared by two or more words

```
# Let's find the shared context between "monstrous" and "very" in  
# Jane Austen's text.
```

Script

Text we  
want to use

The view  
we want

The words we want to find shared  
context for

↓ ↓ ↓  
`text2.common_contexts("monstrous", "very")`



The screenshot shows an IPython console window with a tab labeled 'Console 1/A'. The input is `In [11]: text2.common_contexts(["monstrous", "very"])` and the output is `a_pretty am_glad a_lucky is_pretty be_glad`. An orange arrow points from the explanatory text below to the output of the console.

Here are the words that appear  
around the two words we specified

# Overview

---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions



# Plotting word dispersion

- What if we want to see how a particular word appears across a text? We can use `dispersion_plot` to visualize it.

```
# In order to produce the visualization, make sure that matplotlib
# and numpy are installed.
import matplotlib as ml
import numpy as np

# What is text4?
text4

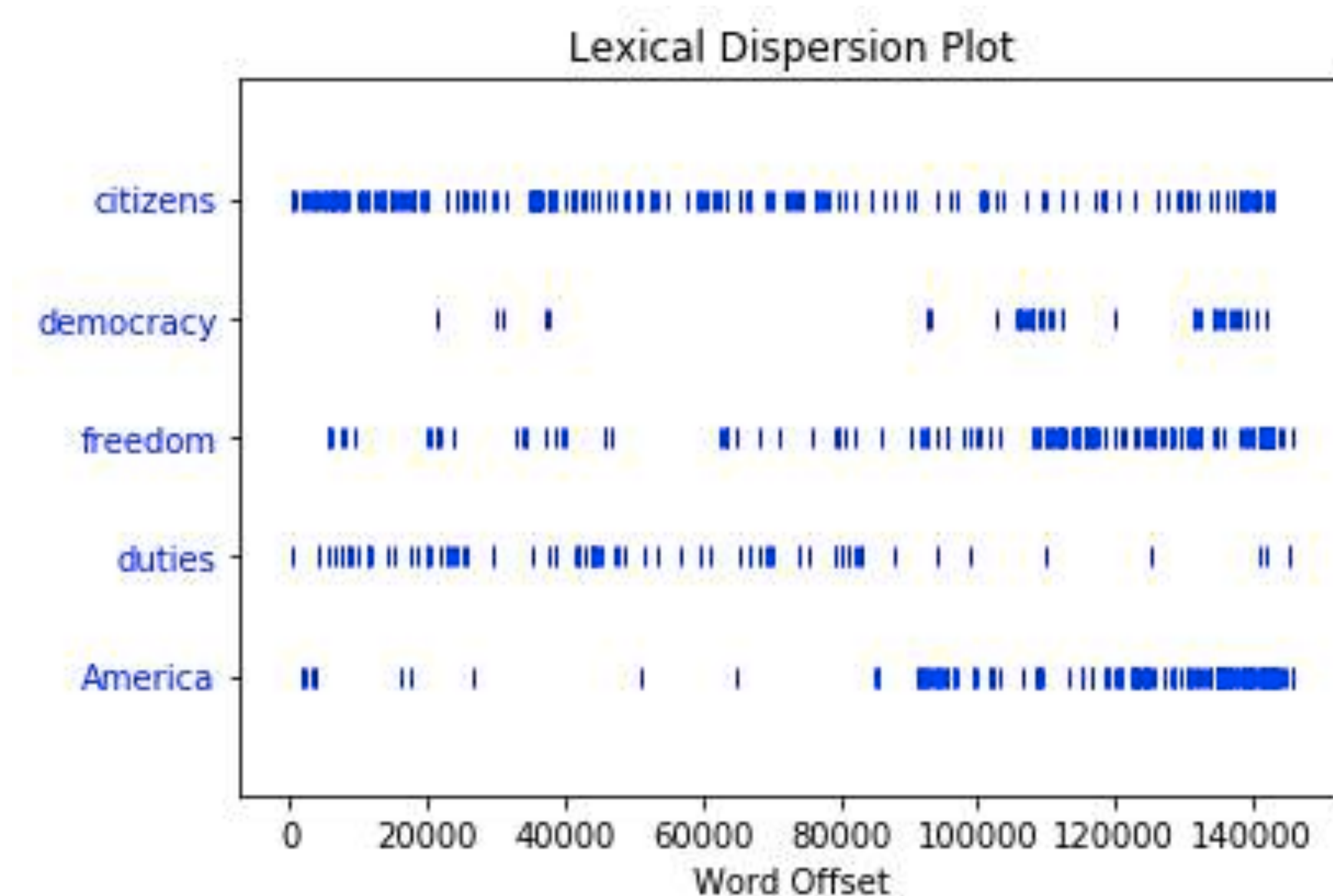
# <Text: Inaugural Address Corpus>
```

Text we want to use      The plot we want      The words we want to visualize in the text

```
text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
```

Script

# Plotting word dispersion



Each line is an instance of the word, and each row represents the entire text

# Investigating vocabulary

- Let's find some metrics of a text

```
# What is text3?  
text3  
  
# <Text: The Book of Genesis>  
  
# Use len() to find the length of the text  
len(text3)  
# 44764
```

This means there are 44,764 words and punctuation symbols, or "tokens", in this text.

Script

*A token is the term for a sequence of characters that we want to treat as a group. We can treat each word as a token, but it does not distinguish between duplicate words – what if we want to find distinct words and symbols?*



# Investigating vocabulary

- In order to find the unique words, we can use the `set("text")` notation.

```
# If we want to find the distinct words and symbols, we can use set("text")
set(text3)

# But this is not very organized - if we wrap that in sorted(), the words
# and symbols will appear alphabetically
sorted(set(text3))

# We can find out the number of unique tokens with len()
len(set(text3))

# 2789
```

Script

This gives us the number of unique tokens (symbols and words)

# Investigating vocabulary

---

- How diverse is our text?

```
# How diverse is our text? We can figure it out by dividing the number of
# unique words by the total number of words.
text6

# <Text: Monty Python and the Holy Grail>

lex_diversity = len(set(text6)) / len(text6)

# Now we can print the answer - we are printing it as a percentage by
# multiplying it by 100 and adding a % sign at the end.
print("The lexical diversity is: ", lex_diversity * 100, "%")
```

Script



# Investigating vocabulary

---

- How diverse is our text?

```
# Let's look up a particular word in the text
text6.count("father")

# 7

# What is the percentage of 'the' in the text?
text6.count('the') / len(text6) * 100

# 1.76%
```

Script

# Investigating vocabulary

Script

```
# If we want to calculate the lexical diversity of any text, we can  
# create a function for it!
```

```
def lex_divers(text):  
    lex_diversity = len(set(text)) / len(text)  
    return print("The lexical diversity is: ",  
                lex_diversity * 100, "%")
```

```
# Here we'll add round() to our calculation so the number looks cleaner. The  
# 2 after the comma is the number of decimal places we want to round to.
```

```
def percent(text, word):  
    text_count = round((text.count(word) / len(text) * 100), 2)  
    return print("The word, ", word, ", appears in ",  
                text, text_count, "% of the text")
```

Tells Python to round the result to 2 decimal places

```
# Let's test out our functions!
```

```
lex_divers(text9)
```

```
percent(text5, "the")
```



```
IPython console  
Console 1/A  
The lexical diversity is: 9.83485761345412 %  
The word, the , appears in <Text: Chat Corpus> 1.44 % of the text
```



# Working with words in lists

---

- A list is a common way to store text in Python

```
# We can create lists with the [ ]  
syntaxsent1 = ['Call', 'me', 'Ishmael', '.']
```

Script

```
# Let's check the length and lexical diversity.  
len(sent1)  
lex_divers(sent1)
```

```
# You can look at some more lists that were imported by typing  
# in sent1 through sent9.  
sent5
```

# Combining words in lists

- We can adjust lists similar to how we learned previously

```
# We can add two lists together with +  
['Monty', 'Python'] + ['and', 'the', 'Holy', 'Grail']
```

Script

```
# Combine variables that represent lists  
sent1 + sent4
```

```
# Append lists with append()  
sent1.append("No")  
sent1
```

← Adds another term  
to the end of a list

```
# ['Call', 'me', 'Ishmael', '.', 'No']
```



# Indexing words

---

- We can adjust lists similar to how we learned previously

```
# We're pulling out the 563rd term from text4.  
text4[563]
```

Script

```
# 'revolution'
```

```
# We're using the index() syntax to return the position of the term  
# we highlighted from text4.  
text4.index('revolution')  
# 563
```

```
# Pull out a specific range of terms  
text6[1501:1525]
```

*Remember! Python is a zero-index language, meaning that the first word has an index of zero.*

# Indexing words

- We can adjust lists similar to how we learned previously

*# If you want all the words up to a certain point, omit the first number.*

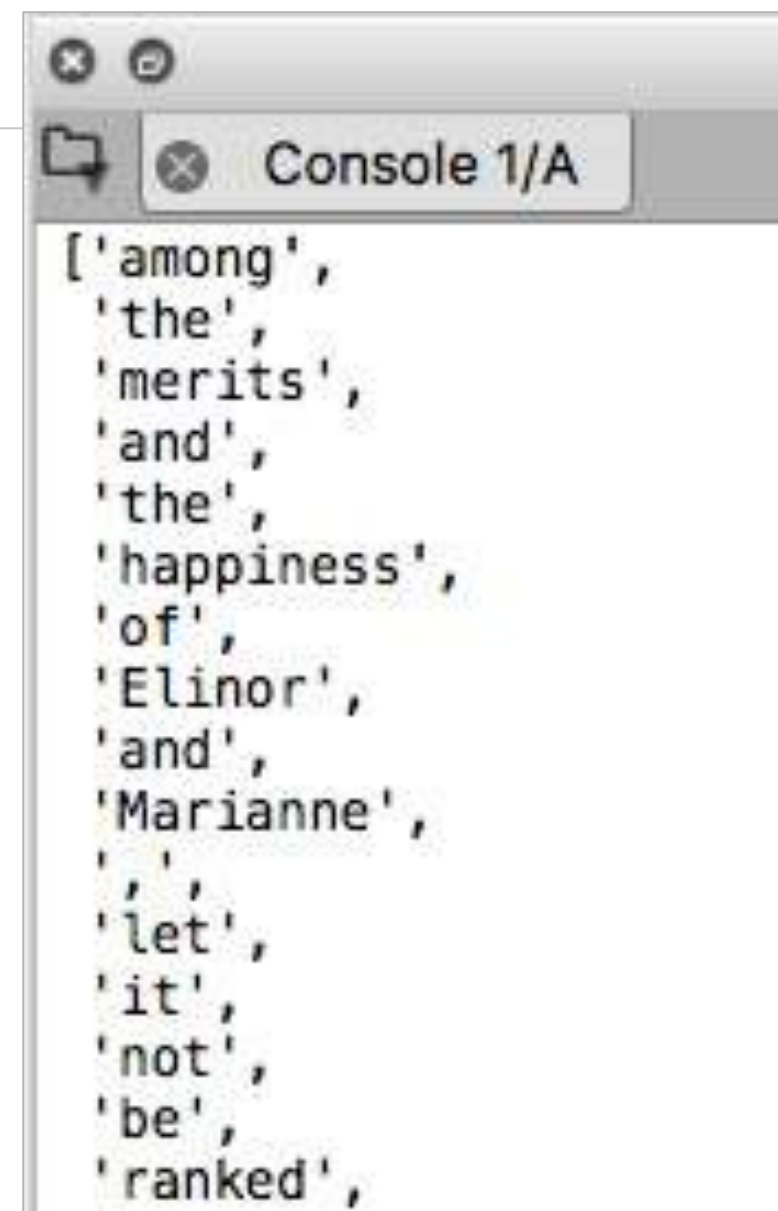
```
text2[:7]
```

```
# ['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen']
```

*# If you want all the words after a certain point, omit the second number.*

```
text2[141525:]
```

Script



A screenshot of a console window titled 'Console 1/A'. It displays a list of words in single quotes, separated by commas and newlines. The words are: 'among', 'the', 'merits', 'and', 'the', 'happiness', 'of', 'Elinor', 'and', 'Marianne', 'let', 'it', 'not', 'be', 'ranked'.

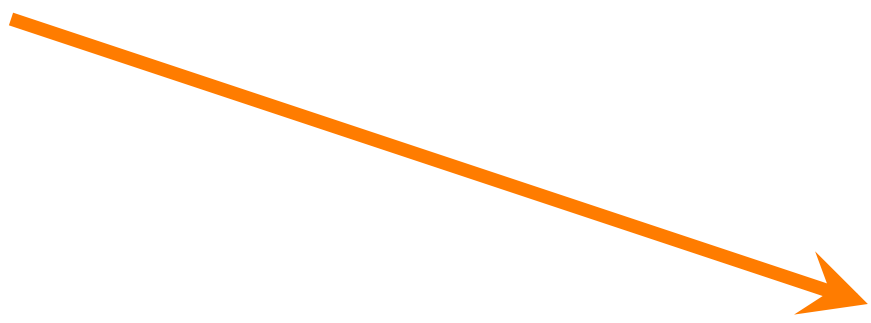


# Sorting words

- Python has a few quirks when we sort through words

```
# First, let's create a list and name it 'my_sent'  
my_sent = ['Bravely', 'bold', 'Sir', 'Robin', ',', 'rode',  
           'forth', 'from', 'Camelot', '.']  
  
# Python sorts words by punctuation first, then uppercase, then lowercase.  
phrase = my_sent[2:5]  
word_sort = sorted(phrase)  
word_sort
```

Script



Console 1/A  
Out[104]: [',', 'Robin', 'Sir']

# Working with strings

- Text is usually read in as a string

*# Operations with strings*

```
name = 'Monty'
```

```
name[0]
```

```
name[:3]
```

*# Operations with strings*

```
name * 2
```

```
name + '!'
```

*# Join two words together with join()*

```
' '.join(['Monty', 'Python'])
```

*# Separate words with split()*

```
'Monty Python'.split()
```

Script



The image shows an IPython console window titled 'IPython console' with a sub-tab 'Console 1/A'. It displays the execution of the code from the script on the left. The code is executed line by line, and the output is shown for each line. The code and its output are as follows:

```
In [108]: name[0]
Out[108]: 'M'

In [109]: name[:3]
Out[109]: 'Mon'

In [110]: name * 2
Out[110]: 'MontyMonty'

In [111]: name + '!'
Out[111]: 'Monty!'

In [112]: ' '.join(['Monty', 'Python'])
Out[112]: 'Monty Python'

In [113]: 'Monty Python'.split()
Out[113]: ['Monty', 'Python']
```

Two orange arrows point from the code in the script to the corresponding lines in the console output. One arrow points from the `' '.join(['Monty', 'Python'])` line to the `In [112]:` line, and the other points from the `'Monty Python'.split()` line to the `In [113]:` line.



# Frequency distribution

- Let's calculate some metrics about the text

```
# We can use FreqDist() to determine the frequency distribution of text1.  
fdist1 = FreqDist(text1)  
print(fdist1)
```

Script

```
# <FreqDist with 19317 samples and 260819 outcomes>
```

Number of unique  
terms in the document

Length of the  
document

```
# What are the 50 most common words in Moby Dick?
```

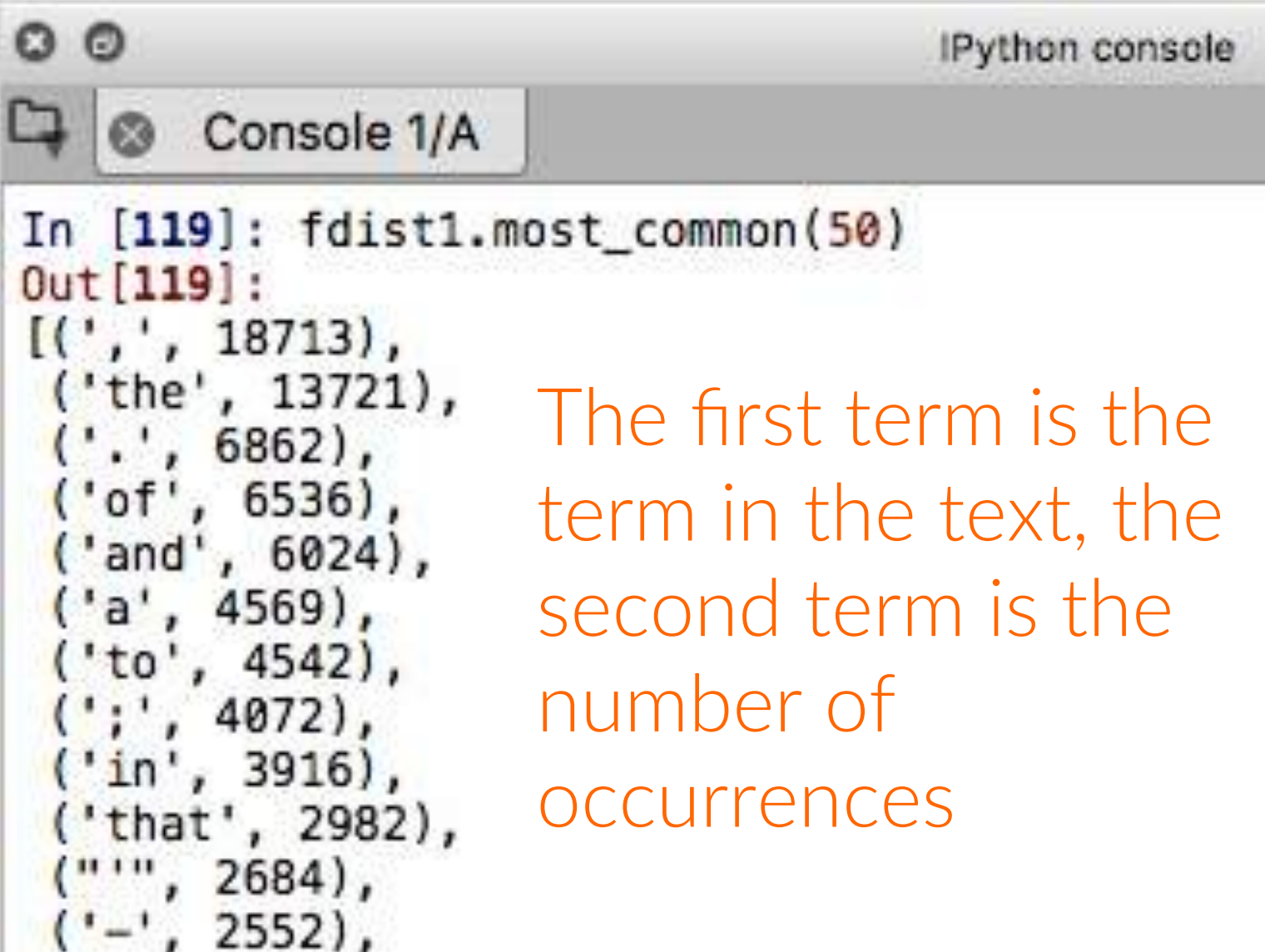
```
fdist1.most_common(50)
```

```
# We can pull out the frequency of a particular word.
```

```
fdist1['whale']
```

This is one of the only more  
distinguishable words!

```
# 906
```



```
IPython console  
Console 1/A  
In [119]: fdist1.most_common(50)  
Out[119]:  
[(',', 18713),  
 ('the', 13721),  
 ('.', 6862),  
 ('of', 6536),  
 ('and', 6024),  
 ('a', 4569),  
 ('to', 4542),  
 (';', 4072),  
 ('in', 3916),  
 ('that', 2982),  
 ('"', 2684),  
 ('-', 2552),
```

The first term is the  
term in the text, the  
second term is the  
number of  
occurrences

# Frequency distribution plot

- We can visualize the distribution of words easily with the plot() syntax

```
# What does the distribution of words look like?
```

```
fdist1.plot(50, cumulative = False)
```

```
# Let's create a cumulative plot of the fifty most popular words
```

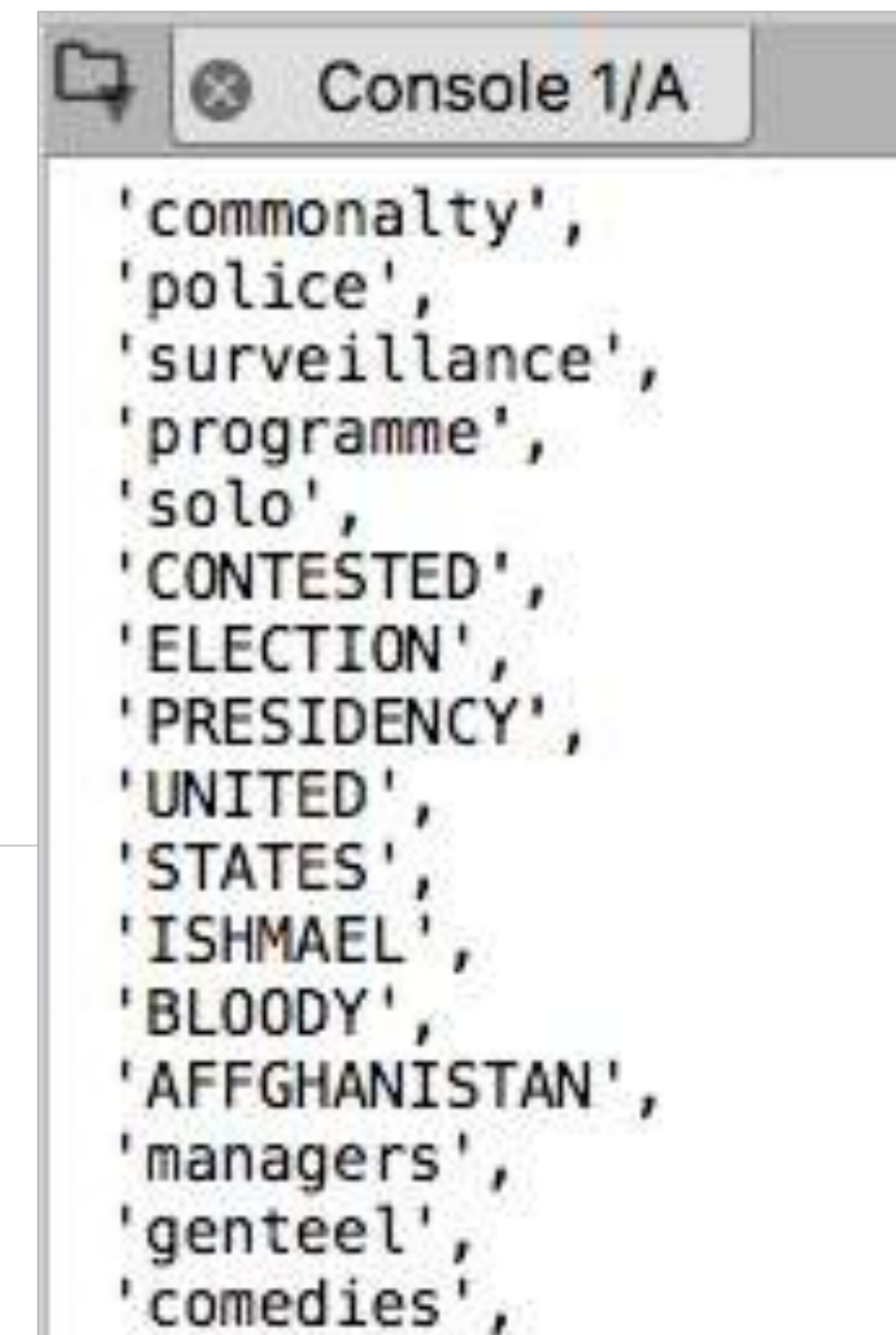
```
fdist1.plot(50, cumulative = True)
```

```
# These 50 words comprise almost 50% of the book!
```

```
# We can look for hapaxes as well - these
```

```
# are words that only occur once.
```

```
fdist1.hapaxes()
```

A screenshot of a console window titled "Console 1/A" showing a list of words in single quotes. An orange arrow points from the `fdist1.hapaxes()` code line to this console output.

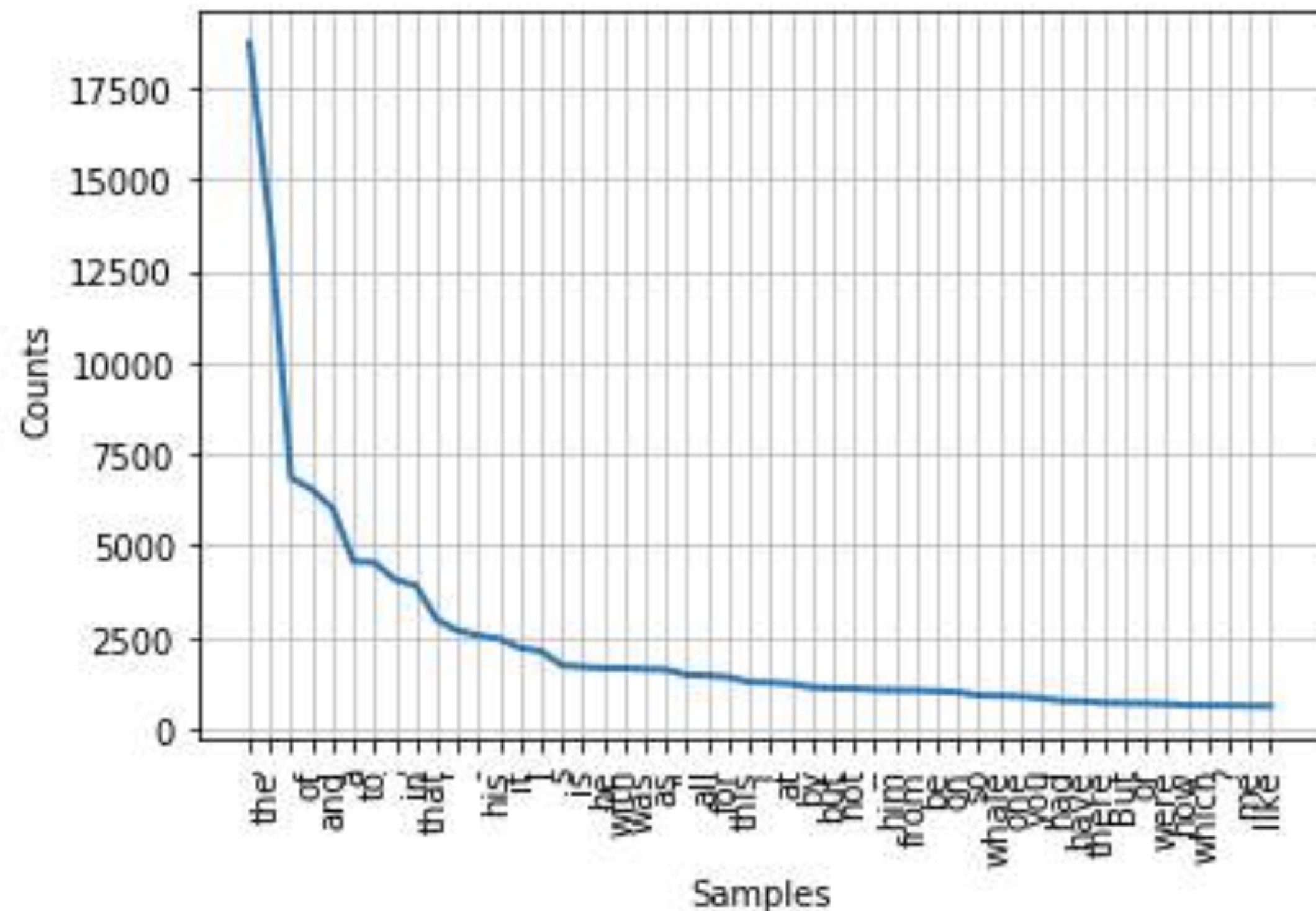
```
'commonalty',  
'police',  
'surveillance',  
'programme',  
'solo',  
'CONTESTED',  
'ELECTION',  
'PRESIDENCY',  
'UNITED',  
'STATES',  
'ISHMAEL',  
'BLOODY',  
'AFFGHANISTAN',  
'managers',  
'genteel',  
'comedies',
```



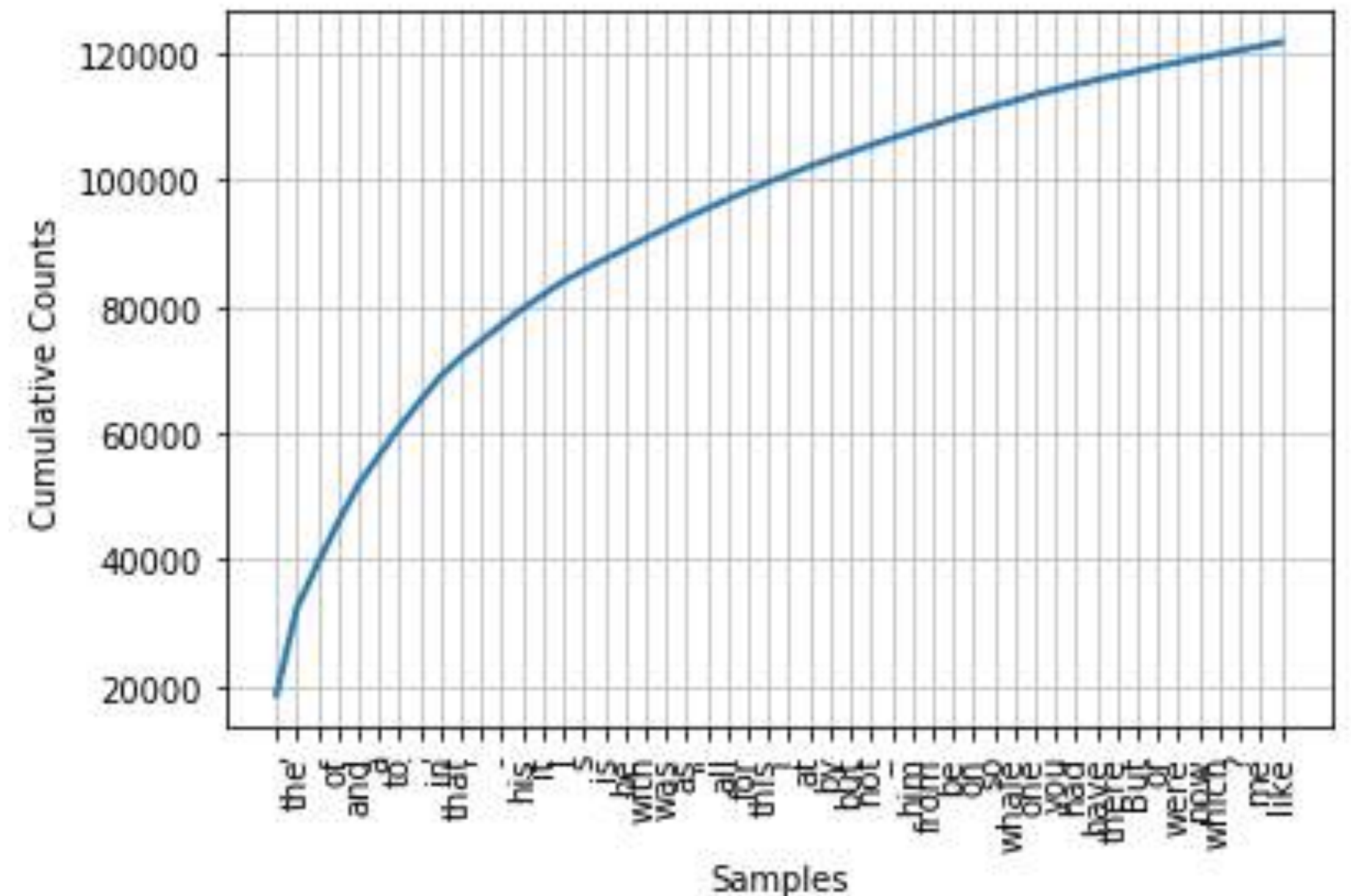
# Frequency distribution plot

- We can visualize the distribution of words easily with the plot() syntax

Not cumulative



Cumulative



*These 50 words comprise almost 50% of the book!*

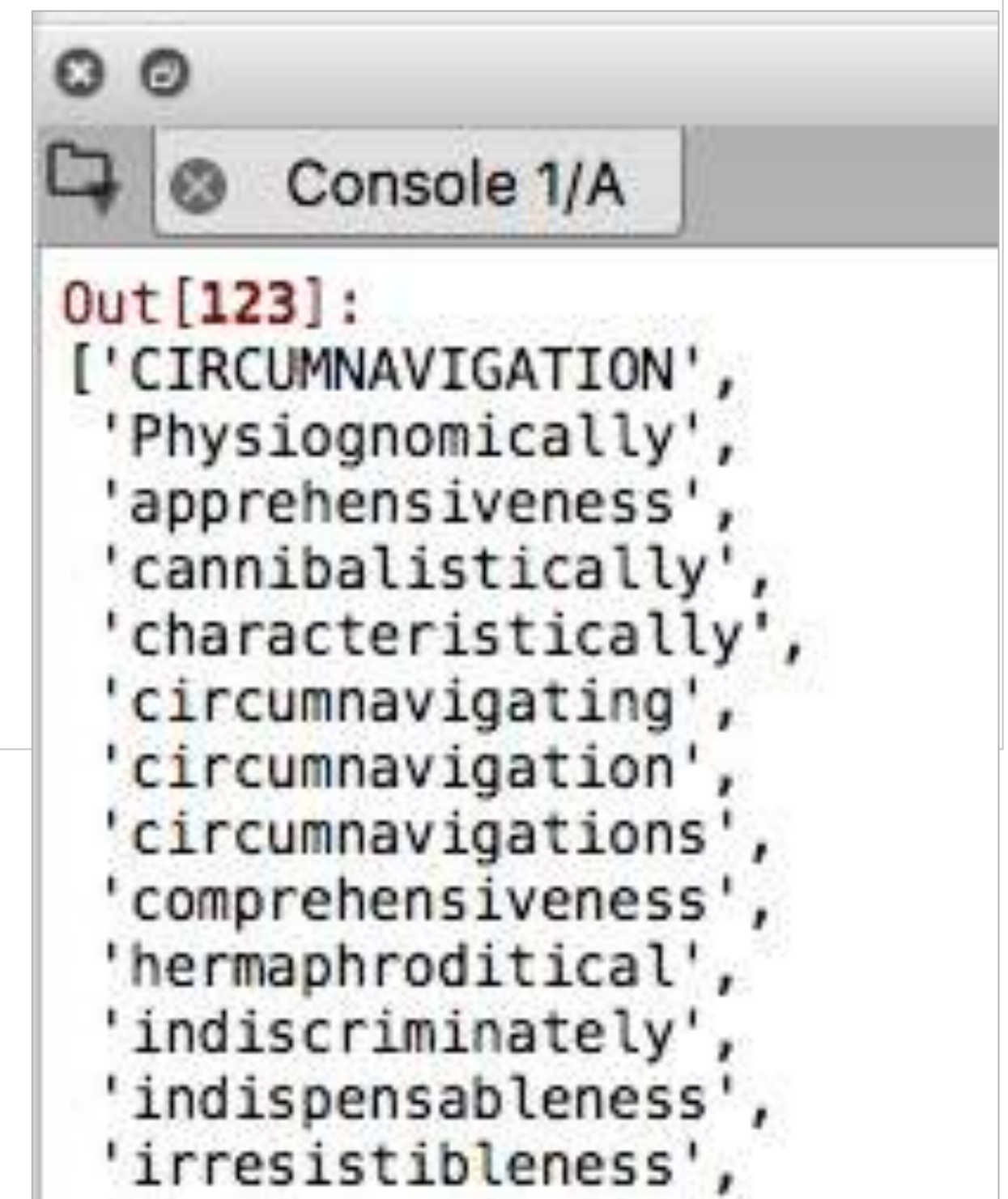
# Subsetting words

- How can we identify words with particular parameters?

```
# What if we want to find words that contain particular constraints?  
# Let's use a for loop to pull out all words with a length greater than 15.  
V = set(text1)
```

Only include the words if the length of the word is greater than 15

```
long_words = [w for w in V if len(w) > 15]  
sorted(long_words)
```



Console 1/A

```
Out[123]:  
['CIRCUMNAVIGATION',  
'Physiognomically',  
'apprehensiveness',  
'cannibalistically',  
'characteristically',  
'circumnavigating',  
'circumnavigation',  
'circumnavigations',  
'comprehensiveness',  
'hermaphroditical',  
'indiscriminately',  
'indispensableness',  
'irresistibleness',
```



# Subsetting words

- How can we identify words with particular parameters?

```
# What if we want to find long words that are frequently occurring?  
# NLTK provides built-in support for this with FreqDist().  
fdist2 = FreqDist(text2)
```

Only include the words if the length of the word is greater than 7 and it occurs more than 7 times in the distribution

```
sorted(w for w in set(text2) if len(w) > 7 and fdist2[w] > 7)
```



```
IPython console  
Console 1/A  
Out [125]:  
['Allenhams',  
'Berkeley',  
'Certainly',  
'Charlotte',  
'Cleveland',  
'Dashwood',  
'Dashwoods',  
'Delafield',  
'Devonshire',  
'Jennings',  
'Longstaple',  
'Margaret',  
'Marianne',  
'Middleton',  
'Middletons',  
'Somersetshire',  
'Whatever',  
'Willoughby',  
'abilities',  
'absolutely',  
'acknowledge',  
'acknowledged',  
'acquaintance',  
'acquainted',  
'admiration',  
'advanced',
```

# Overview

---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions

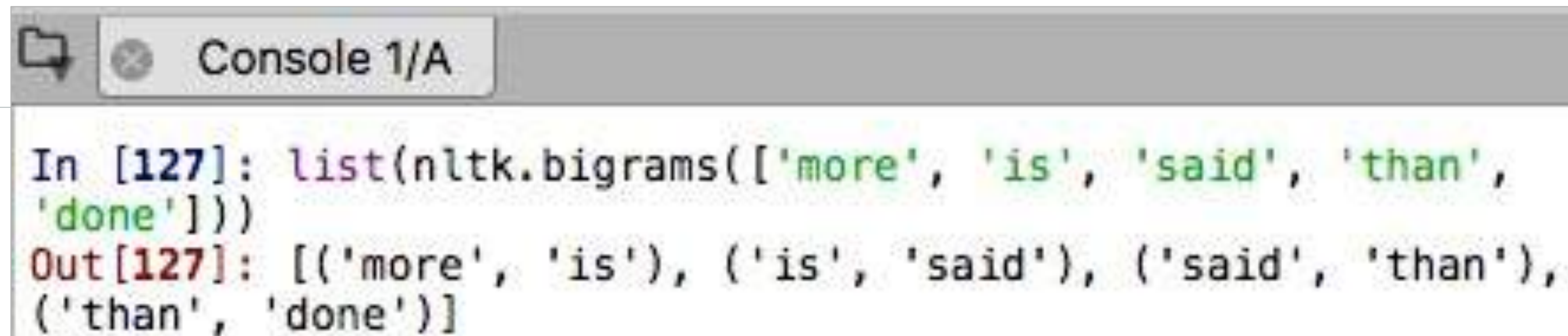


# Finding collocations

- How can we identify words with particular parameters?

```
# We can use the bigrams() function to extract word pairs that  
# occur together unusually often.  
list(nltk.bigrams(['more', 'is', 'said', 'than', 'done']))
```

Script



The screenshot shows a Jupyter Notebook interface with a tab labeled "Console 1/A". The input code is: `In [127]: list(nltk.bigrams(['more', 'is', 'said', 'than', 'done']))`. The output is: `Out[127]: [('more', 'is'), ('is', 'said'), ('said', 'than'), ('than', 'done')]`.

*Note: if you don't include list() in nltk, you will get an output that doesn't make sense. We need to tell Python to convert it into a list*

# Finding bigrams

- A **collocation** is a sequence of words that occur together unusually often

```
# Collocations are sequences of words that occur unusually often  
# and contain rare words.  
# What are the most common word occurrences  
# in an Inaugural Address?  
text4.collocations()
```

Script

```
# What about a corpus that contains  
# personal ads?  
text8.collocations()
```

An IPython console window titled "IPython console" with a sub-tab "Console 1/A". It displays the output of the command `text4.collocations()`. The output lists various word pairs: "United States; fellow citizens; four years; years ago; Federal Government; General Government; American people; Vice President; Old World; Almighty God; Fellow citizens; Chief Magistrate; Chief Justice; God bless; every citizen; Indian tribes; public debt; one another; foreign nations; political parties".

```
IPython console  
Console 1/A  
In [129]: text4.collocations()  
United States; fellow citizens; four years; years ago; Federal  
Government; General Government; American people; Vice President;  
Old  
World; Almighty God; Fellow citizens; Chief Magistrate; Chief  
Justice;  
God bless; every citizen; Indian tribes; public debt; one another;  
foreign nations; political parties
```

An IPython console window titled "IPython console" with a sub-tab "Console 1/A". It displays the output of the command `text8.collocations()`. The output lists various word pairs: "would like; medium build; social drinker; quiet nights; non smoker; long term; age open; Would like; easy going; financially secure; fun times; similar interests; Age open; weekends away; poss rship; well presented; never married; single mum; permanent relationship; slim build".

```
IPython console  
Console 1/A  
In [130]: text8.collocations()  
would like; medium build; social drinker; quiet nights; non  
smoker;  
long term; age open; Would like; easy going; financially secure;  
fun  
times; similar interests; Age open; weekends away; poss rship;  
well  
presented; never married; single mum; permanent relationship; slim  
build
```



# Counting word lengths

- We can also find out other metrics, such as word lengths

```
# Return the lengths of each word in text1.
```

```
[len(w) for w in text1]
```

```
# [1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, ...]
```

```
# Now, wrap that code in FreqDist to see the  
# frequency distribution of word lengths.
```

```
fdist = FreqDist(len(w) for w in text1)
```

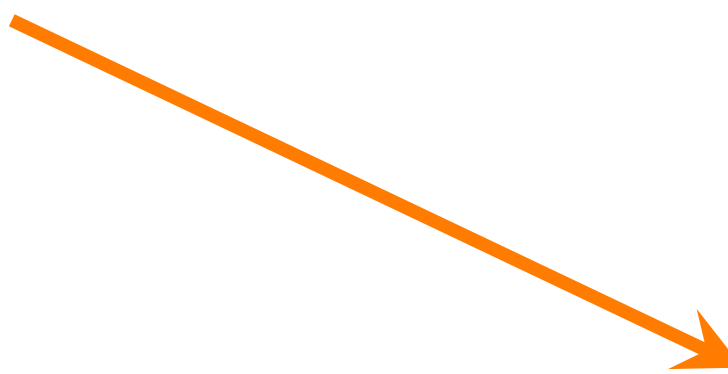
```
print(fdist)
```

```
fdist
```

```
# <FreqDist with 19 samples and 260819 outcomes>
```

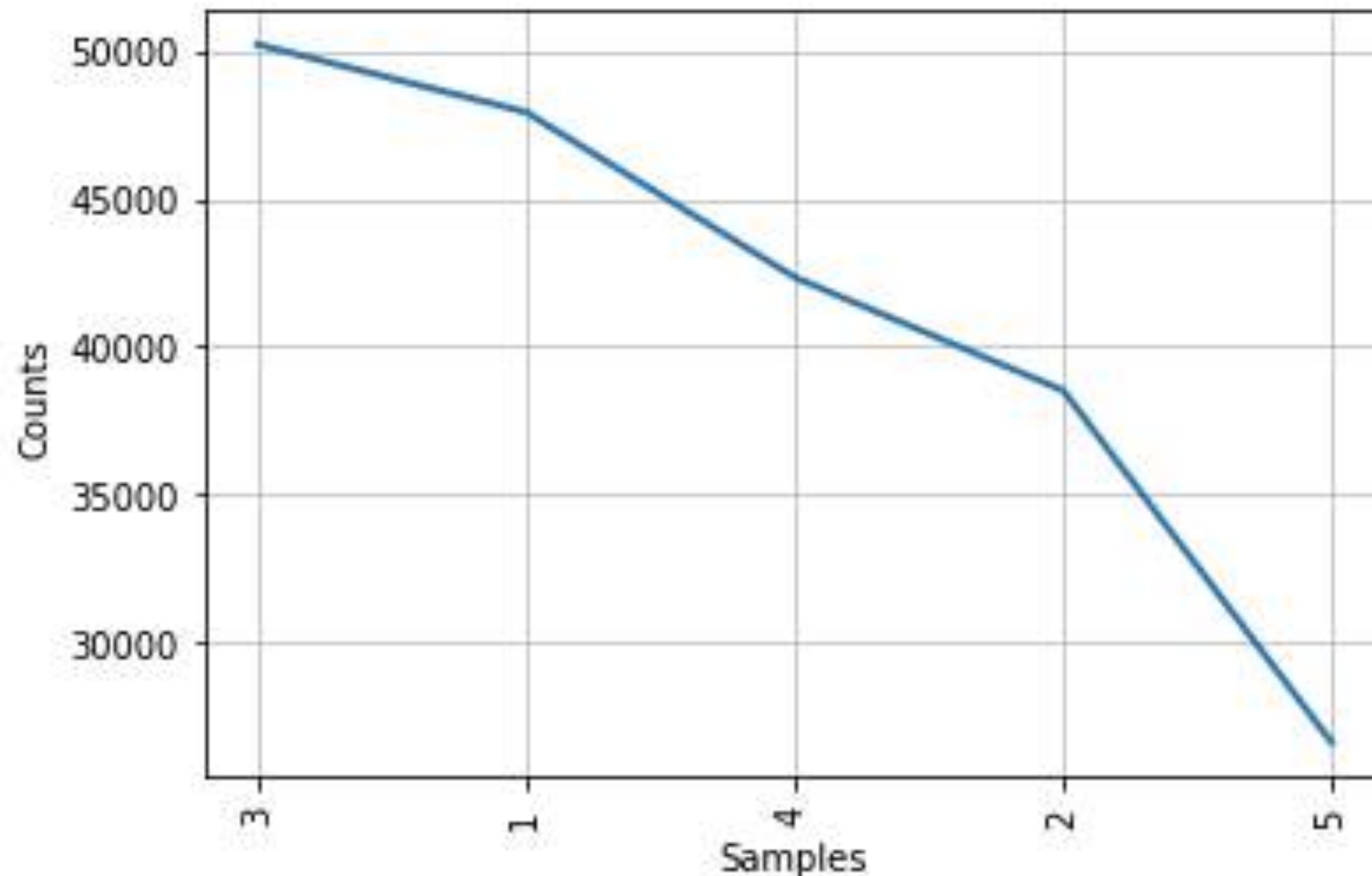
```
# Plot the top 5 frequently occurring word lengths
```

```
fdist.plot(5)
```



```
Console 1/A  
  
In [133]: fdist  
Out[133]:  
FreqDist({1: 47933,  
          2: 38513,  
          3: 50223,  
          4: 42345,  
          5: 26597,  
          6: 17111,  
          7: 14399,  
          8: 9966,  
          9: 6428
```

# Plotting word dispersion



There is a high concentration of 3-letter and 1-letter words



# Counting word lengths

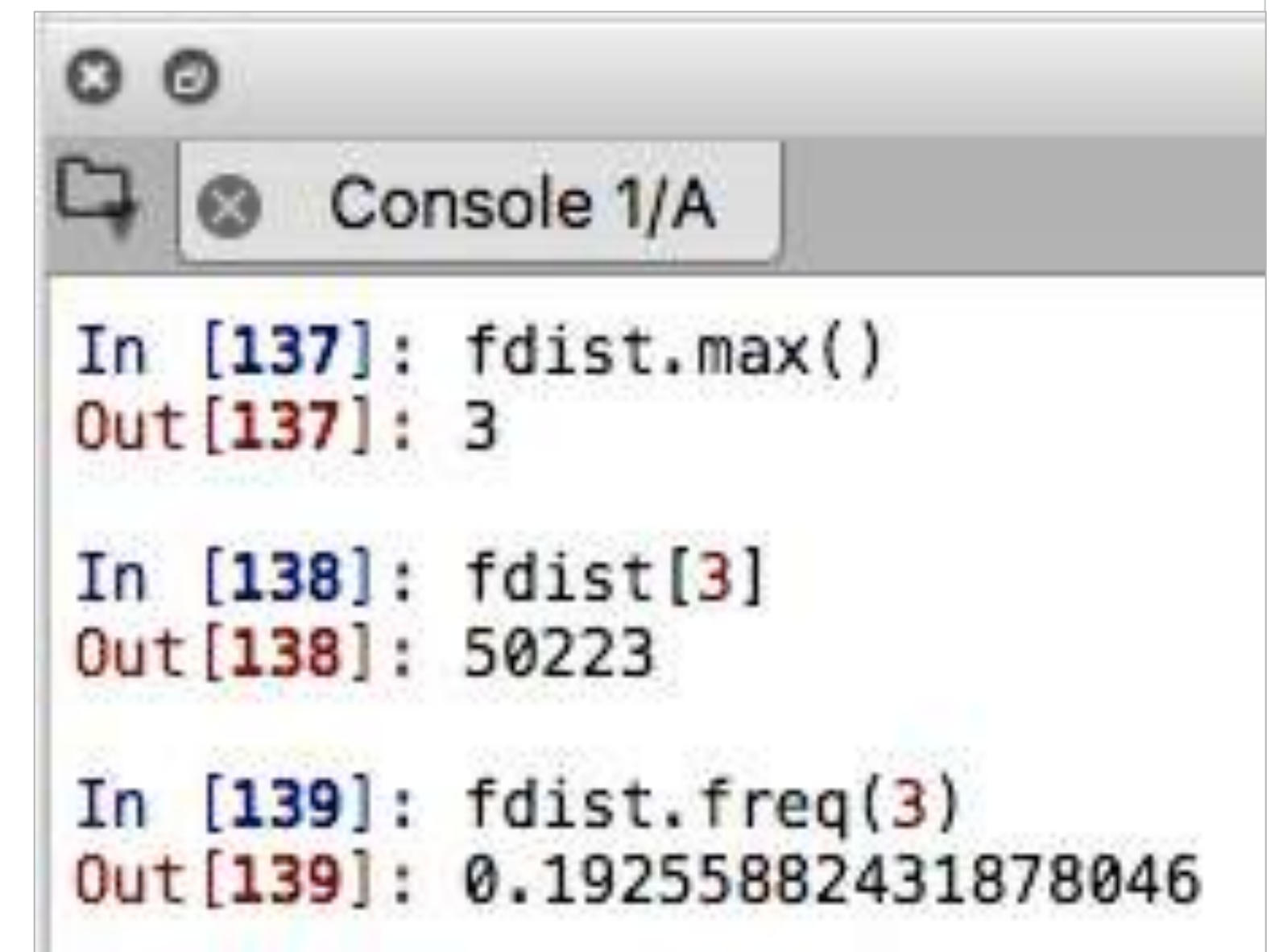
- We can also find out other metrics, such as word lengths

```
# We're using the most_common() syntax to pull out the most  
# common occurrences of fdist.  
fdist.most_common()
```

```
# The max() syntax identifies the term that has the highest occurrence.  
fdist.max()  
# 3
```

```
# Here, we're asking how many occurrences  
# there were for the term '3'.  
fdist[3]  
# 50223
```

```
# What is the percent of 3-letter words in the text?  
fdist.freq(3)  
# 0.19255882431878046
```



```
Console 1/A  
In [137]: fdist.max()  
Out[137]: 3  
  
In [138]: fdist[3]  
Out[138]: 50223  
  
In [139]: fdist.freq(3)  
Out[139]: 0.19255882431878046
```

# Functions for frequency distributions

Example	Description
<code>fdist = FreqDist(samples)</code>	create a frequency distribution containing the given samples
<code>fdist[sample] += 1</code>	increment the count for this sample
<code>fdist['monstrous']</code>	count of the number of times a given sample occurred
<code>fdist.freq('monstrous')</code>	frequency of a given sample
<code>fdist.N()</code>	total number of samples
<code>fdist.most_common(n)</code>	the n most common samples and their frequencies
<code>for sample in fdist:</code>	iterate over the samples
<code>fdist.max()</code>	sample with the greatest count
<code>fdist.tabulate()</code>	tabulate the frequency distribution
<code>fdist.plot()</code>	graphical plot of the frequency distribution
<code>fdist.plot(cumulative=True)</code>	cumulative plot of the frequency distribution
<code>fdist1  = fdist2</code>	update fdist1 with counts from fdist2
<code>fdist1 &lt; fdist2</code>	test if samples in fdist1 occur less frequently than in fdist2



# Overview

---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions

# Word comparison operators

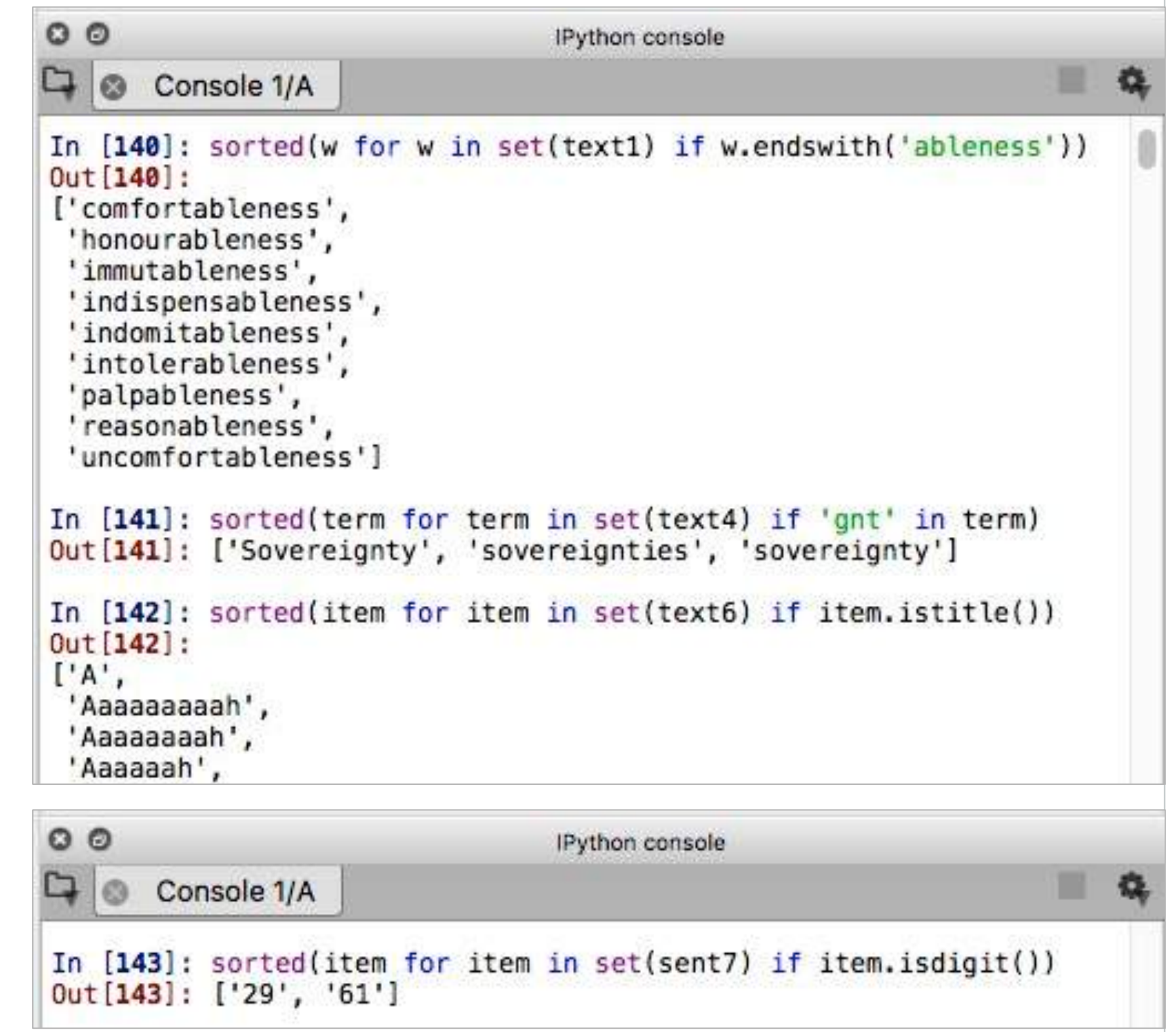
- We can also look for particular characteristics in the corpus

```
# Find all the words in text1 that end in 'ableness'.  
sorted(w for w in set(text1) if w.endswith('ableness'))
```

```
# Find all the words in text4 that contain 'gnt'.  
sorted(term for term in set(text4)  
if 'gnt' in term)
```

```
# Find all the words in text6 if words are  
# capitalized.  
sorted(item for item in set(text6)  
if item.istitle())
```

```
# Find all the terms in sent7 that are numbers.  
sorted(item for item in set(sent7)  
if item.isdigit())
```



```
IPython console  
Console 1/A  
In [140]: sorted(w for w in set(text1) if w.endswith('ableness'))  
Out[140]: ['comfortableness',  
'honourableness',  
'immutableness',  
'indispensableness',  
'indomitableness',  
'intolerableness',  
'palpableness',  
'reasonableness',  
'uncomfortableness']  
  
In [141]: sorted(term for term in set(text4) if 'gnt' in term)  
Out[141]: ['Sovereignty', 'sovereignties', 'sovereignty']  
  
In [142]: sorted(item for item in set(text6) if item.istitle())  
Out[142]: ['A',  
'Aaaaaaaaah',  
'Aaaaaaaaah',  
'Aaaaaah']  
  
IPython console  
Console 1/A  
In [143]: sorted(item for item in set(sent7) if item.isdigit())  
Out[143]: ['29', '61']
```



# Word comparison operators

---

Example	Description
<code>s.startswith(t)</code>	test if s starts with t
<code>s.endswith(t)</code>	test if s ends with t
<code>t in s</code>	test if t is a substring of s
<code>s.islower()</code>	test if s contains cased characters and all are lowercase
<code>s.isupper()</code>	test if s contains cased characters and all are uppercase
<code>s.isalpha()</code>	test if s is non-empty and all characters in s are alphabetic
<code>s.isalnum()</code>	test if s is non-empty and all characters in s are alphanumeric
<code>s.isdigit()</code>	test if s is non-empty and all characters in s are digits
<code>s.istitle()</code>	test if s contains cased characters and is titlecased (i.e. all words in s have initial capitals)

# More complex operators

---

- We can also add multiple parameters to identify words

```
# Find all the words in text7 that contain '-' and 'index'.
sorted(w for w in set(text7) if '-' in w and 'index' in w)

# Find all the words in text3 that are capitalized with a
# length greater than 10.
sorted(wd for wd in set(text3) if wd.istitle() and len(wd) > 10)

# Find all the words in text7 if words are not all lowercase.
sorted(w for w in set(text7) if not w.islower())

# Find all the terms in text2 that have either 'cie' or 'cei'.
sorted(t for t in set(text2) if 'cie' in t or 'cei' in t)
```



# Word comparison operators

```
IPython console
Console 1/A

In [146]: sorted(w for w in set(text7) if '-' in w and 'index' in w)
Out[146]:
['Stock-index',
 'index-arbitrage',
 'index-fund',
 'index-options',
 'index-related',
 'stock-index']

In [147]: sorted(wd for wd in set(text3) if wd.istitle() and len(wd) > 10)
Out[147]:
['Abelmizraim',
 'Allonbachuth',
 'Beerlahairoi',
```

```
IPython console
Console 1/A

In [148]: sorted(w for w in set(sent7) if not w.islower())
Out[148]:
['.', '29', '61', 'Nov.', 'Pierre', 'Vinken']

In [149]: sorted(t for t in set(text2) if 'cie' in t or 'cei' in t)
Out[149]:
['ancient',
 'ceiling',
 'conceit',
 'conceited',
 'conceive',
 'conscience',
 'conscientious',
 'conscientiously',
 'deceitful',
 'deceive',
```

# Dealing with word cases

- Earlier, we counted up all the unique terms, but it didn't distinguish between capitalized and non-capitalized words that were the same  
I.e. "This" and "this" were counted as two separate words

```
# Eliminate double counting of the same words that  
# have different cases - here, we're setting all the words  
# to lower case, then only counting the alphabetic terms.  
len(set(word.lower() for word in text1 if word.isalpha()))  
  
# 16948
```

← This is the number of unique words in the text!

Script

# Looping with conditions

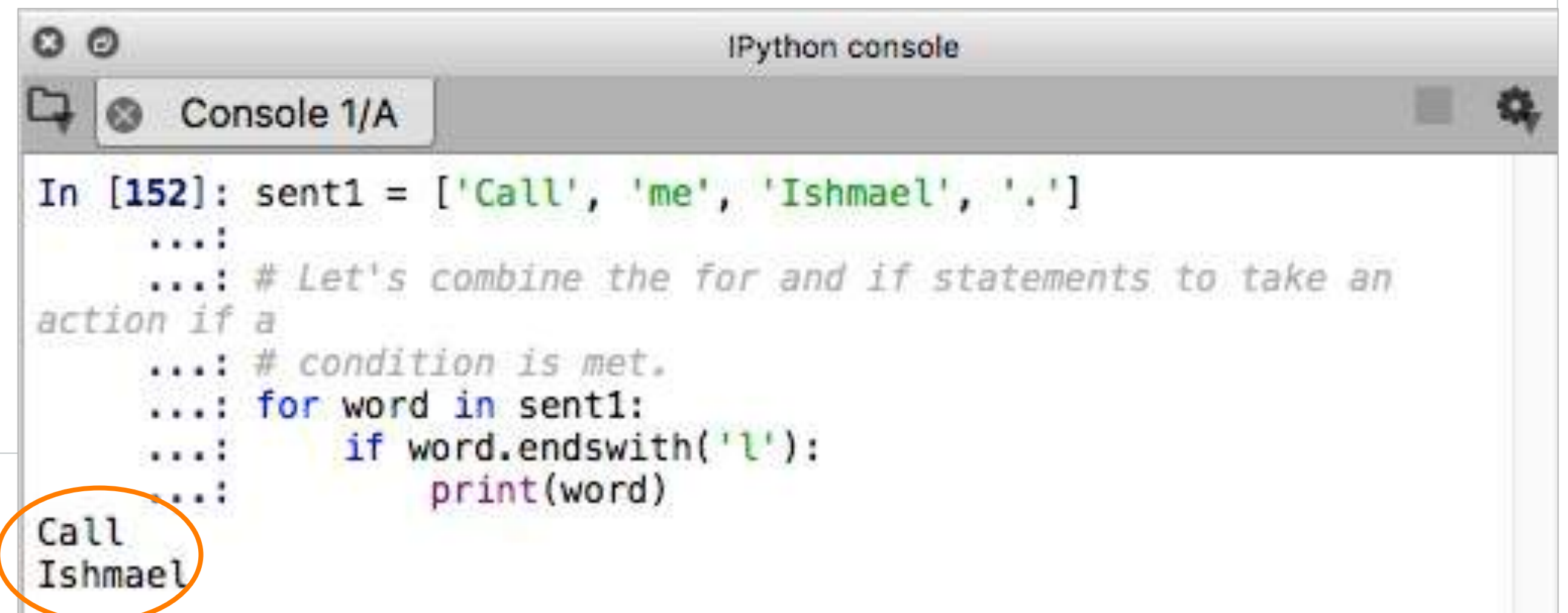
- We can combine conditionals to pull out particular words from text

```
sent1 = ['Call', 'me', 'Ishmael', '.']
```

Script

```
# Let's combine the for and if statements to identify  
# all words that end in 'l'.
```

```
for word in sent1:  
    if word.endswith('l'):  
        print(word)
```



The image shows an IPython console window titled 'IPython console' with a sub-tab 'Console 1/A'. It displays the execution of the code from the previous block. The input is 'In [152]: sent1 = ['Call', 'me', 'Ishmael', '.']'. The output shows the words 'Call' and 'Ishmael' printed on separate lines. An orange arrow points from the 'print(word)' line in the code block to the output in the console. The words 'Call' and 'Ishmael' are circled in orange in the console output.

```
IPython console  
Console 1/A  
In [152]: sent1 = ['Call', 'me', 'Ishmael', '.']  
...:  
...: # Let's combine the for and if statements to take an  
action if a  
...: # condition is met.  
...: for word in sent1:  
...:     if word.endswith('l'):  
...:         print(word)  
...:  
Call  
Ishmael
```



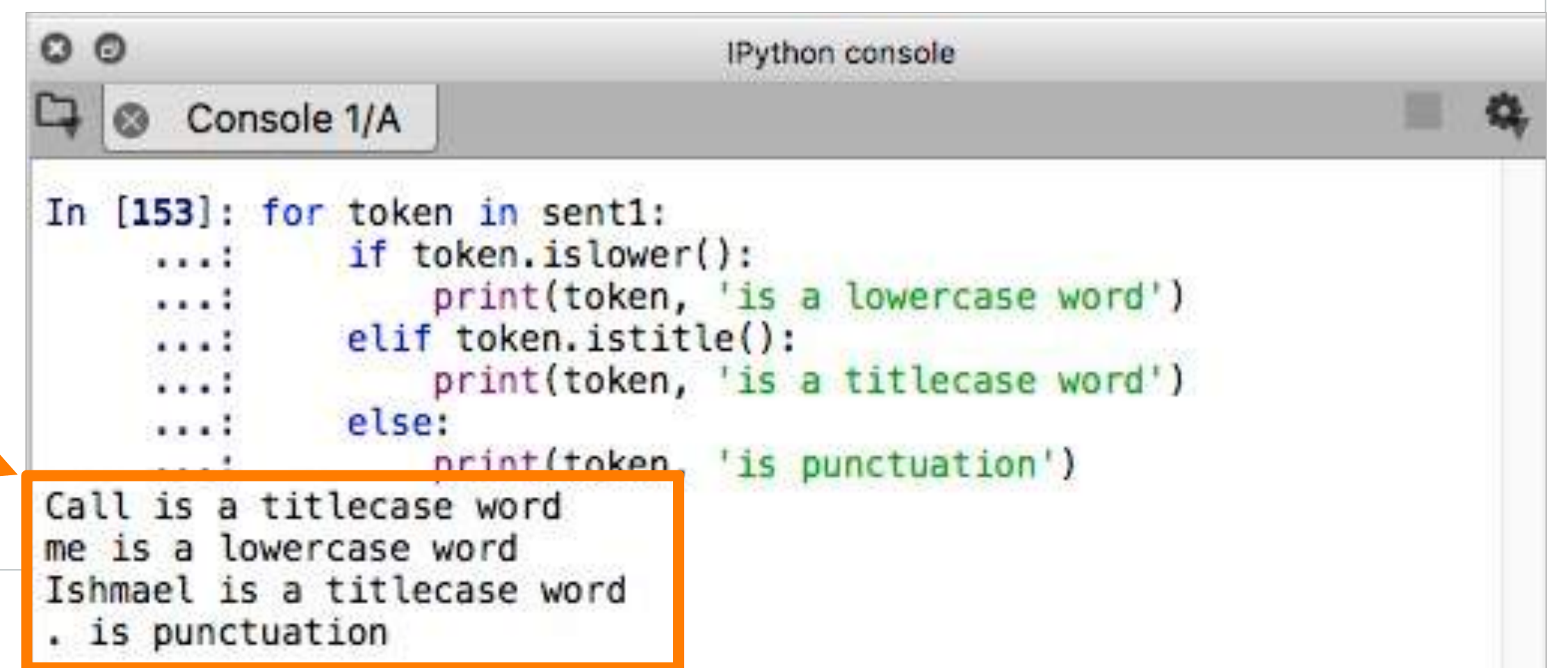
# Looping with conditions

- We can combine conditionals to pull out particular words from text

*# We can print different responses based in the criteria we specify*

Script

```
for word in sent1:
    if token.islower():
        print(token, 'is a lowercase word')
    elif token.istitle():
        print(token, 'is a titlecase word')
    else:
        print(token, 'is punctuation')
```



```
IPython console
Console 1/A

In [153]: for token in sent1:
...:     if token.islower():
...:         print(token, 'is a lowercase word')
...:     elif token.istitle():
...:         print(token, 'is a titlecase word')
...:     else:
...:         print(token, 'is punctuation')

Call is a titlecase word
me is a lowercase word
Ishmael is a titlecase word
. is punctuation
```

# Project Gutenberg

- This project offers over 54,000 free eBooks that you can download
- The NLTK package includes a selection of texts from there, which we will use to work with text





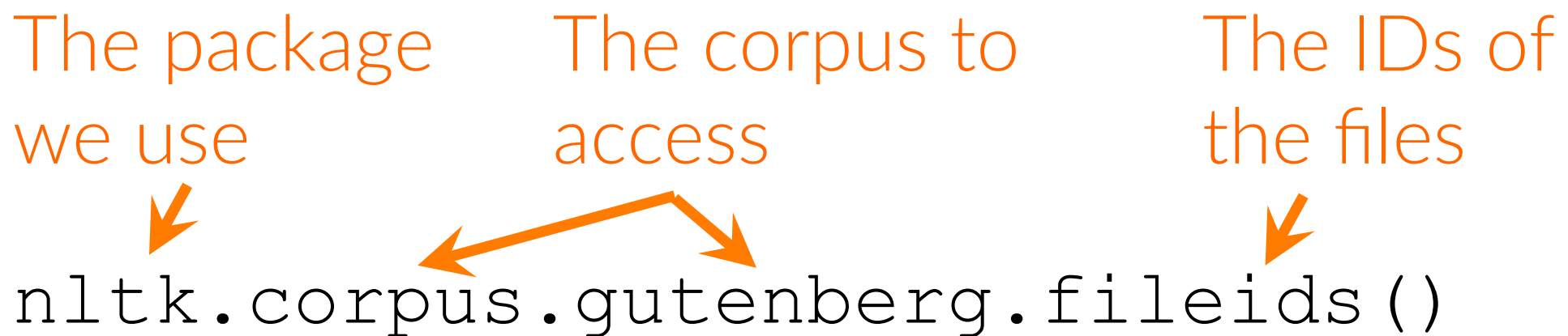
# Importing larger texts

- Here we can access the Gutenberg texts

*# We can access the Gutenberg texts with the following code:*

Script

The package we use      The corpus to access      The IDs of the files



```
nltk.corpus.gutenberg.fileids()
```

*# Let's pick out the first of these texts – Emma by Jane Austen – and give it a short name, emma, then find out how many words it contains:*

```
emma = nltk.corpus.gutenberg.words('austen-emma.txt')  
len(emma)
```

*# 192427* ← Length of the text



# Importing larger texts

- Earlier, we used `text1.concordance` to find words in a text – but this assumed that we were using one of the texts we imported from the book import
- Since we're now using texts from `nltk.corpus`, we have to use the following statements to perform the same tasks as before

```
# First, we can assign the text to a shorter variable, 'emma'.
```

Script

The package  
we use

The corpus to  
access

Pull out  
the text

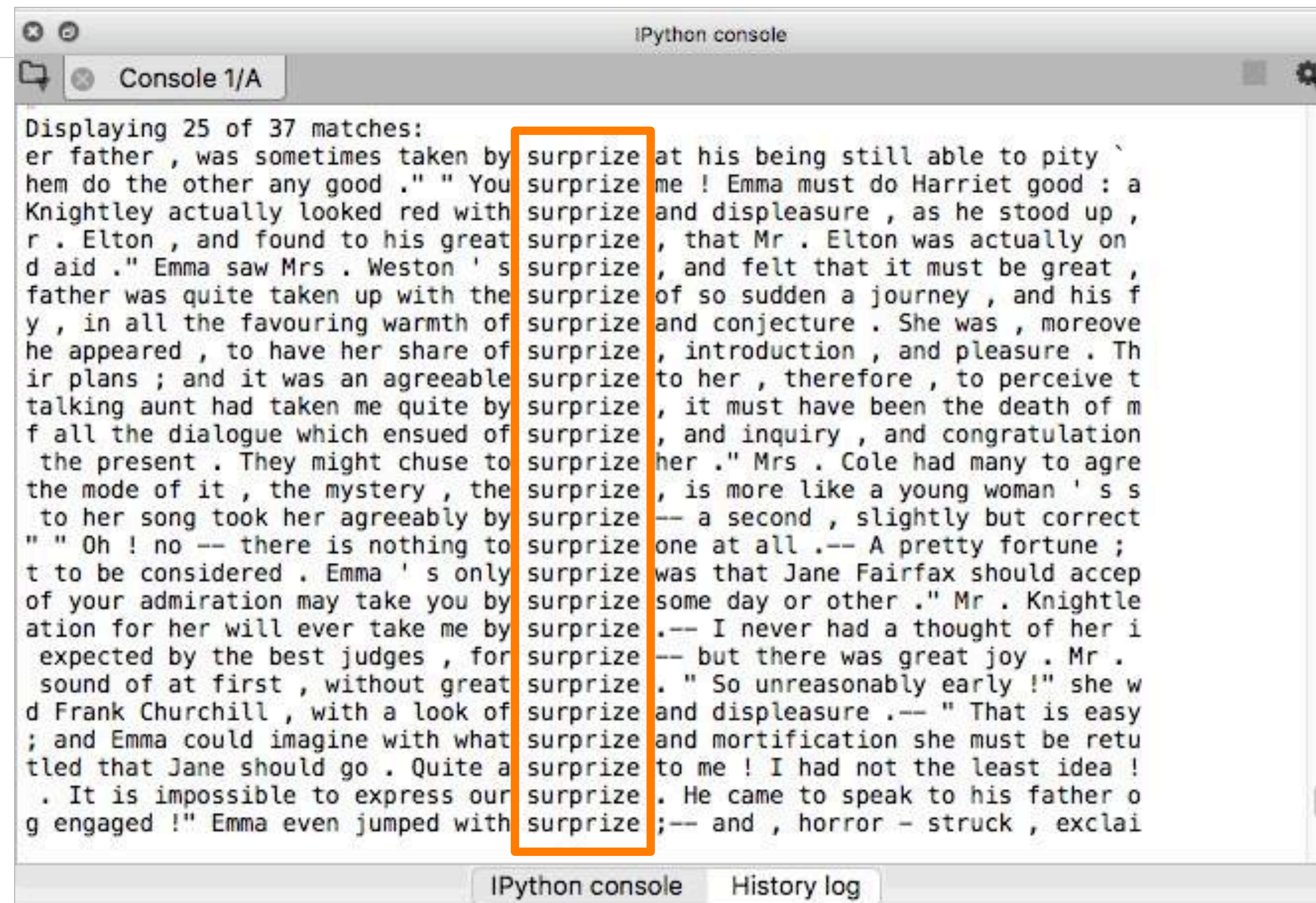
```
emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))
```



# Importing larger texts

```
# Now, we can use concordance to find the words around 'surprise'  
emma.concordance("surprise")
```

Script



```
IPython console  
Console 1/A  
Displaying 25 of 37 matches:  
er father , was sometimes taken by surprise at his being still able to pity `  
hem do the other any good ." " You surprise me ! Emma must do Harriet good : a  
Knightley actually looked red with surprise and displeasure , as he stood up ,  
r . Elton , and found to his great surprise , that Mr . Elton was actually on  
d aid ." Emma saw Mrs . Weston ' s surprise , and felt that it must be great ,  
father was quite taken up with the surprise of so sudden a journey , and his f  
y , in all the favouring warmth of surprise and conjecture . She was , moreove  
he appeared , to have her share of surprise , introduction , and pleasure . Th  
ir plans ; and it was an agreeable surprise to her , therefore , to perceive t  
talking aunt had taken me quite by surprise , it must have been the death of m  
f all the dialogue which ensued of surprise , and inquiry , and congratulation  
the present . They might chuse to surprise her ." Mrs . Cole had many to agre  
the mode of it , the mystery , the surprise , is more like a young woman ' s s  
to her song took her agreeably by surprise -- a second , slightly but correct  
" " Oh ! no -- there is nothing to surprise one at all .-- A pretty fortune ;  
t to be considered . Emma ' s only surprise was that Jane Fairfax should accep  
of your admiration may take you by surprise some day or other ." Mr . Knightle  
ation for her will ever take me by surprise .-- I never had a thought of her i  
expected by the best judges , for surprise -- but there was great joy . Mr .  
sound of at first , without great surprise . " So unreasonably early !" she w  
d Frank Churchill , with a look of surprise and displeasure .-- " That is easy  
; and Emma could imagine with what surprise and mortification she must be retu  
tled that Jane should go . Quite a surprise to me ! I had not the least idea !  
 . It is impossible to express our surprise . He came to speak to his father o  
g engaged !" Emma even jumped with surprise ;-- and , horror - struck , exclai
```



# Importing larger texts

---

- Instead of having to assign a text with that cumbersome syntax, Python provides another way for us to import the text

```
# Import the gutenber texts from nltk.corpus  
from nltk.corpus import gutenber
```

Script

```
# Look at the names of the files  
gutenber.fileids()
```

```
# ['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', ...]
```

```
# Set the variable name 'emma' equal to the text "Emma" by Jane Austen  
emma = gutenber.words('austen-emma.txt')
```



# Automating information

---

- Let's automate a program to view more metrics about each file that we have imported – first, we'll look at the raw data

```
for fileid in gutenbergl.fileids():  
    # The length of the raw file (number of characters)  
    num_chars = len(gutenberg.raw(fileid))  
    # The number of words in the file  
    num_words = len(gutenberg.words(fileid))  
    # The number of sentences in the file  
    num_sents = len(gutenberg.sents(fileid))  
    # The number of unique words in the file  
    num_vocab = len(set(w.lower() for w in gutenbergl.words(fileid)))  
    # Print the metrics we want to see  
    print(fileid, num_chars, num_words, num_sents, num_vocab)
```

Script

# Automating information

---

- What if we want to look at some additional metrics?

```
for fileid in gutenbergl.fileids():  
    # The length of the raw file (number of characters)  
    num_chars = len(gutenberg.raw(fileid))  
    # The number of words in the file  
    num_words = len(gutenberg.words(fileid))  
    # The number of sentences in the file  
    num_sents = len(gutenberg.sents(fileid))  
    # The number of unique words in the file  
    num_vocab = len(set(w.lower() for w in gutenbergl.words(fileid)))  
    # Print the metrics we want to see - the number of characters per word, the  
    # number of words per sentence, and the diversity of vocabulary.  
    print(round(num_chars/num_words), round(num_words/num_sents),  
          round(num_vocab/num_words * 100, 2), "% ", fileid)
```

Script



# Segmenting texts

- We can use `sents()` to find the number of sentences in each text

```
# We can divide up each text into sentences with sents(). We can view  
# the sentences here  
macbeth_sentences = gutenbergsents('shakespeare-macbeth.txt')  
macbeth_sentences  
  
# View a particular sentence with the [ ] syntax  
macbeth_sentences[1116]  
  
# Identify the longest sentence in Macbeth  
longest_len = max(len(s) for s in macbeth_sentences)  
[s for s in macbeth_sentences if len(s) == longest_len]
```

Script

*Most NLTK corpus readers include a variety of access methods apart from `words()`, `raw()`, and `sents()`.*

# Overview

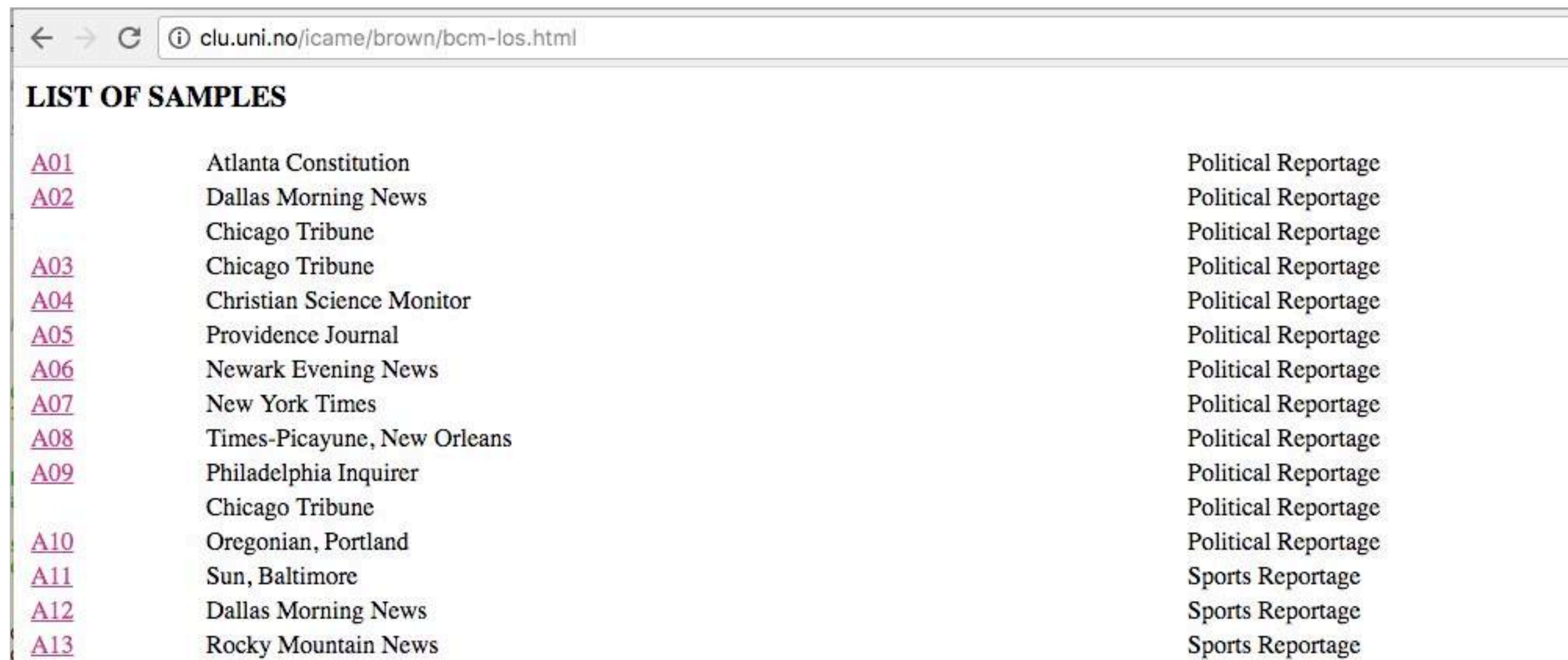
---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions



# The Brown corpus

- This corpus is the first million-word electronic corpus of English – assembled in 1961 at Brown university
- The 500 sources in the corpus have been categorized – you can learn more at <http://clu.uni.no/icame/brown/bcm-los.html>



LIST OF SAMPLES		
<a href="#">A01</a>	Atlanta Constitution	Political Reportage
<a href="#">A02</a>	Dallas Morning News	Political Reportage
	Chicago Tribune	Political Reportage
<a href="#">A03</a>	Chicago Tribune	Political Reportage
<a href="#">A04</a>	Christian Science Monitor	Political Reportage
<a href="#">A05</a>	Providence Journal	Political Reportage
<a href="#">A06</a>	Newark Evening News	Political Reportage
<a href="#">A07</a>	New York Times	Political Reportage
<a href="#">A08</a>	Times-Picayune, New Orleans	Political Reportage
<a href="#">A09</a>	Philadelphia Inquirer	Political Reportage
	Chicago Tribune	Political Reportage
<a href="#">A10</a>	Oregonian, Portland	Political Reportage
<a href="#">A11</a>	Sun, Baltimore	Sports Reportage
<a href="#">A12</a>	Dallas Morning News	Sports Reportage
<a href="#">A13</a>	Rocky Mountain News	Sports Reportage

# The Brown corpus

- Let's investigate the categories in the Brown corpus

*# First, import the Brown corpus and view the categories*

```
from nltk.corpus import brown
brown.categories()
```

*# Which words are in the 'news' category?*

```
brown.words(categories = 'news')
```

*# Which words are in the file id 'cg22'?*

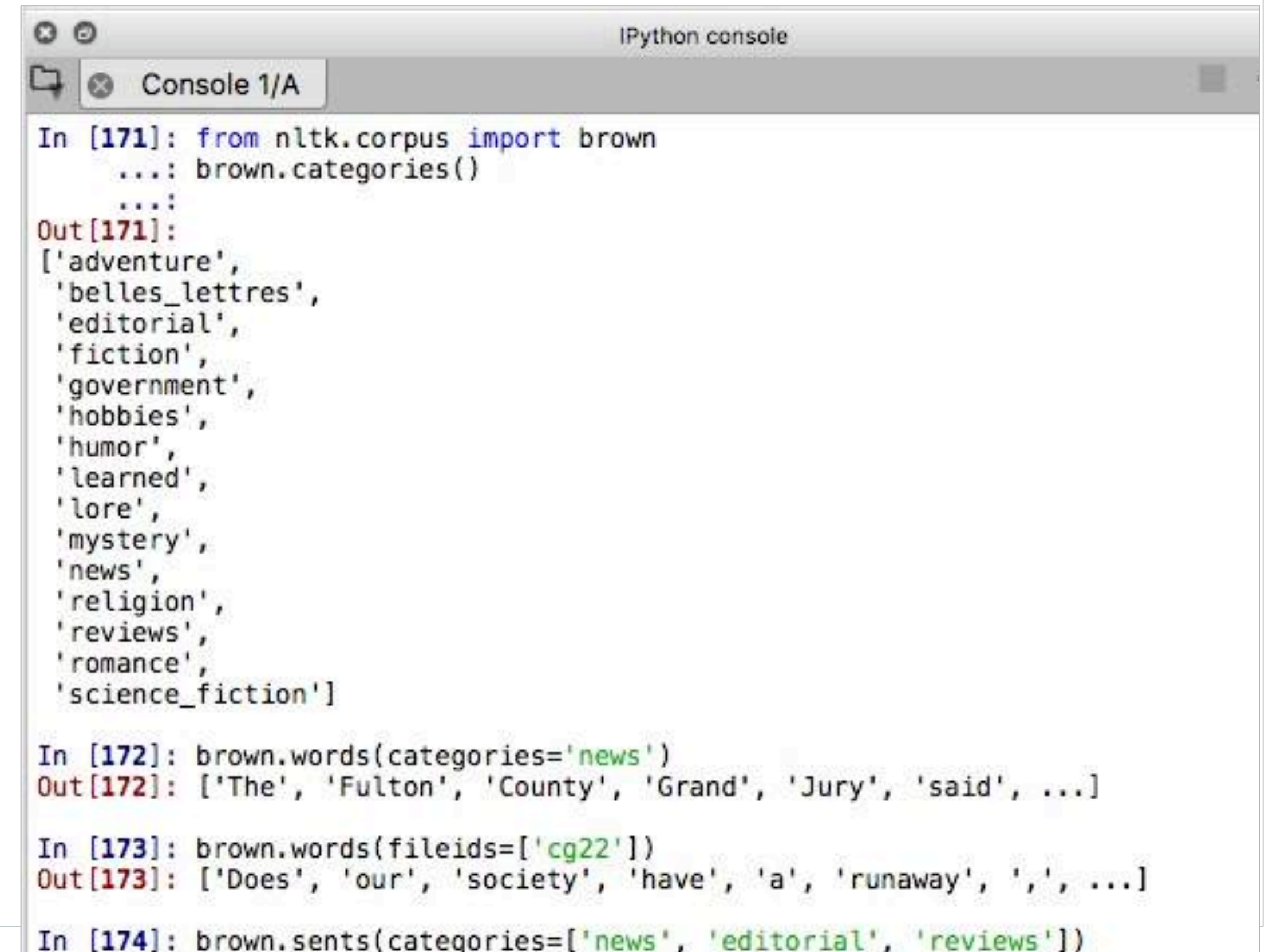
```
brown.words(fileids = ['cg22'])
```

*# Which sentences are in the categories*

*# 'news', 'editorial', and 'reviews'?*

```
brown.sents(categories = ['news',
                          'editorial',
                          'reviews'])
```

Script



```
IPython console
Console 1/A

In [171]: from nltk.corpus import brown
...: brown.categories()
...:
Out[171]:
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']

In [172]: brown.words(categories='news')
Out[172]: ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

In [173]: brown.words(fileids=['cg22'])
Out[173]: ['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]

In [174]: brown.sents(categories=['news', 'editorial', 'reviews'])
```



# The Brown corpus

- How do the genres differ in their usage of verbs? We'll need to create the word counts for a genre

```
# First, import the Brown corpus
from nltk.corpus import brown

# Define the 'news_text' variable as the words in the 'news' category
news_text = brown.words(categories = 'news')

# Find the frequency distribution for the words
# Make sure to use the lower() syntax so that we're not double counting words
fdist = nltk.FreqDist(w.lower() for w in news_text)

# Define the words we want to look at, and then print the distribution
modals = ['can', 'could', 'may', 'might', 'must', 'will']
for m in modals:
    print(m + ': ', fdist[m], end = ' ')
```

Script

We include end=' ' to tell the print function to put its output on a single line.



# The Brown corpus

---

- Now we'll get the counts for each genre we want to use – we'll use the conditional frequency distribution support from NLTK, which is beyond the scope of this course

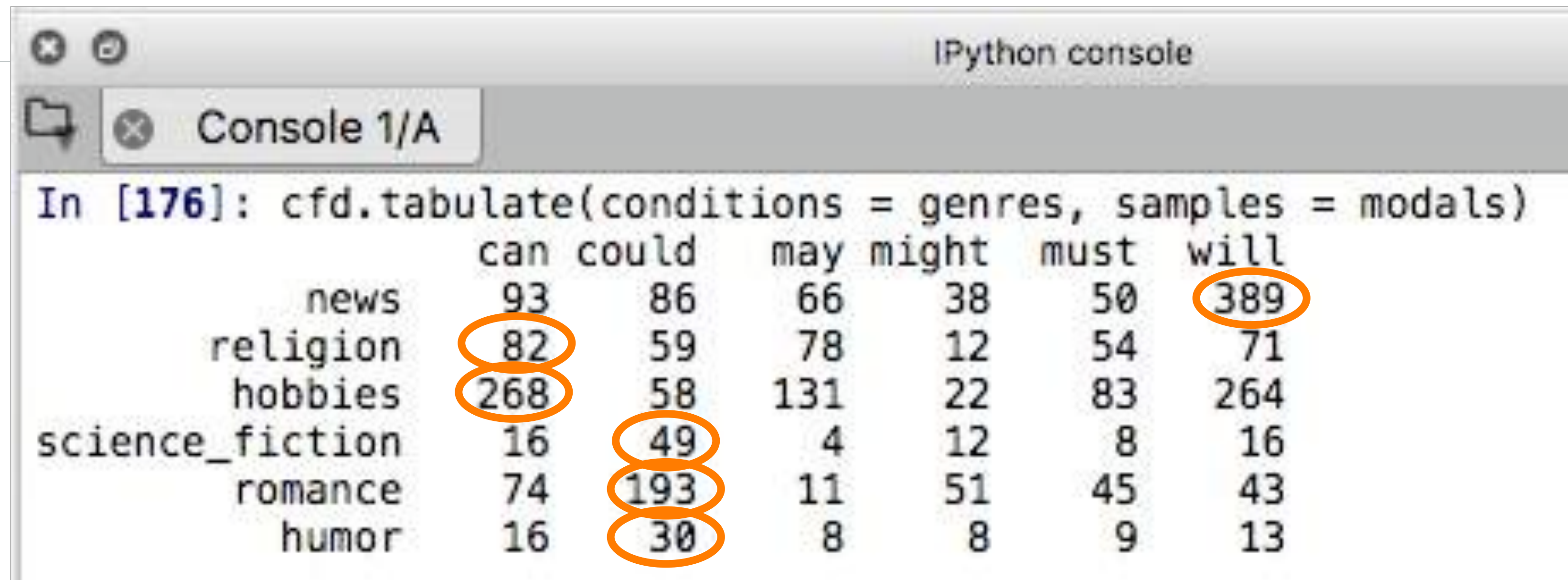
```
# This will tabulate the frequency distributions of the words we specify
# in the genres we specify
cfd = nltk.ConditionalFreqDist(
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories = genre))
genres = ['news', 'religion', 'hobbies', 'science_fiction', 'romance', 'humor']
modals = ['can', 'could', 'may', 'might', 'must', 'will']
```

Script

# The Brown corpus

```
# Create the chart to view the distribution of words across the genre  
cfd.tabulate(conditions = genres, samples = modals)
```

Script



IPython console

Console 1/A

In [176]: cfd.tabulate(conditions = genres, samples = modals)

	can	could	may	might	must	will
news	93	86	66	38	50	389
religion	82	59	78	12	54	71
hobbies	268	58	131	22	83	264
science_fiction	16	49	4	12	8	16
romance	74	193	11	51	45	43
humor	16	30	8	8	9	13

*What are some patterns that you can observe? What other words would you be interested in comparing across genres?*

# The Reuters corpus

- This corpus has over 10,000 news documents with over 1.3 million words
- The texts are classified into 90 topics, and grouped into a 'training' and 'test' set to train algorithms

```
# Import the Reuters corpus
from nltk.corpus import reuters

# Look at the file ids and categories
reuters.fileids()
reuters.categories()
```

Script

A screenshot of an IPython console window. The window has a title bar that says "IPython console" and a tab labeled "Console 1/A". The console shows the command "In [178]: reuters.categories()" and the output "Out[178]:" followed by a list of category names: ['acq', 'alum', 'barley', 'bop', 'carcass', 'castor-oil', 'cocoa', 'coconut', 'coconut-oil', 'coffee', ...].

```
IPython console
Console 1/A
In [178]: reuters.categories()
Out[178]:
['acq',
 'alum',
 'barley',
 'bop',
 'carcass',
 'castor-oil',
 'cocoa',
 'coconut',
 'coconut-oil',
 'coffee',
 ...]
```

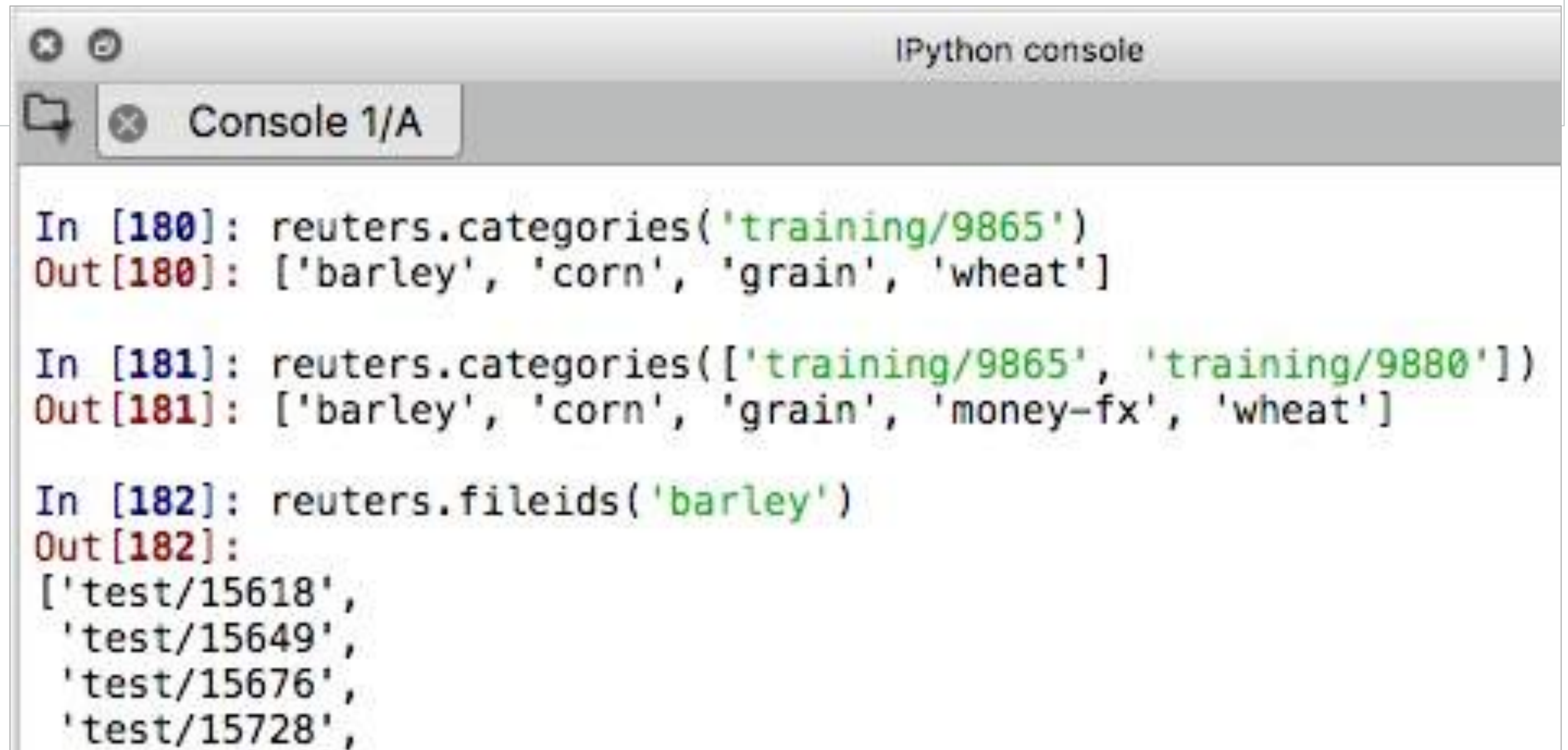


# The Reuters corpus

- Categories in this corpus overlap because articles usually cover multiple topics
  - we can look at overlapping topics by identifying multiple texts

```
# What do some of the texts show?  
reuters.categories('training/9865')  
reuters.categories(['training/9865', 'training/9880'])  
reuters.fileids('barley')  
reuters.fileids(['barley', 'corn'])
```

Script



IPython console

Console 1/A

```
In [180]: reuters.categories('training/9865')  
Out[180]: ['barley', 'corn', 'grain', 'wheat']  
  
In [181]: reuters.categories(['training/9865', 'training/9880'])  
Out[181]: ['barley', 'corn', 'grain', 'money-fx', 'wheat']  
  
In [182]: reuters.fileids('barley')  
Out[182]:  
['test/15618',  
 'test/15649',  
 'test/15676',  
 'test/15728',
```

# The Reuters corpus

---

- We can pull out words or sentences we want from texts with the [ ] syntax
- The first words in each text are the titles, which are upper case

```
# How do we pull out specific words?  
reuters.words('training/9865')[:14]  
reuters.words(['training/9865', 'training/9880'])  
reuters.words(categories = 'barley')  
reuters.words(categories = ['barley', 'corn'])
```

Script

# The Inaugural Address corpus

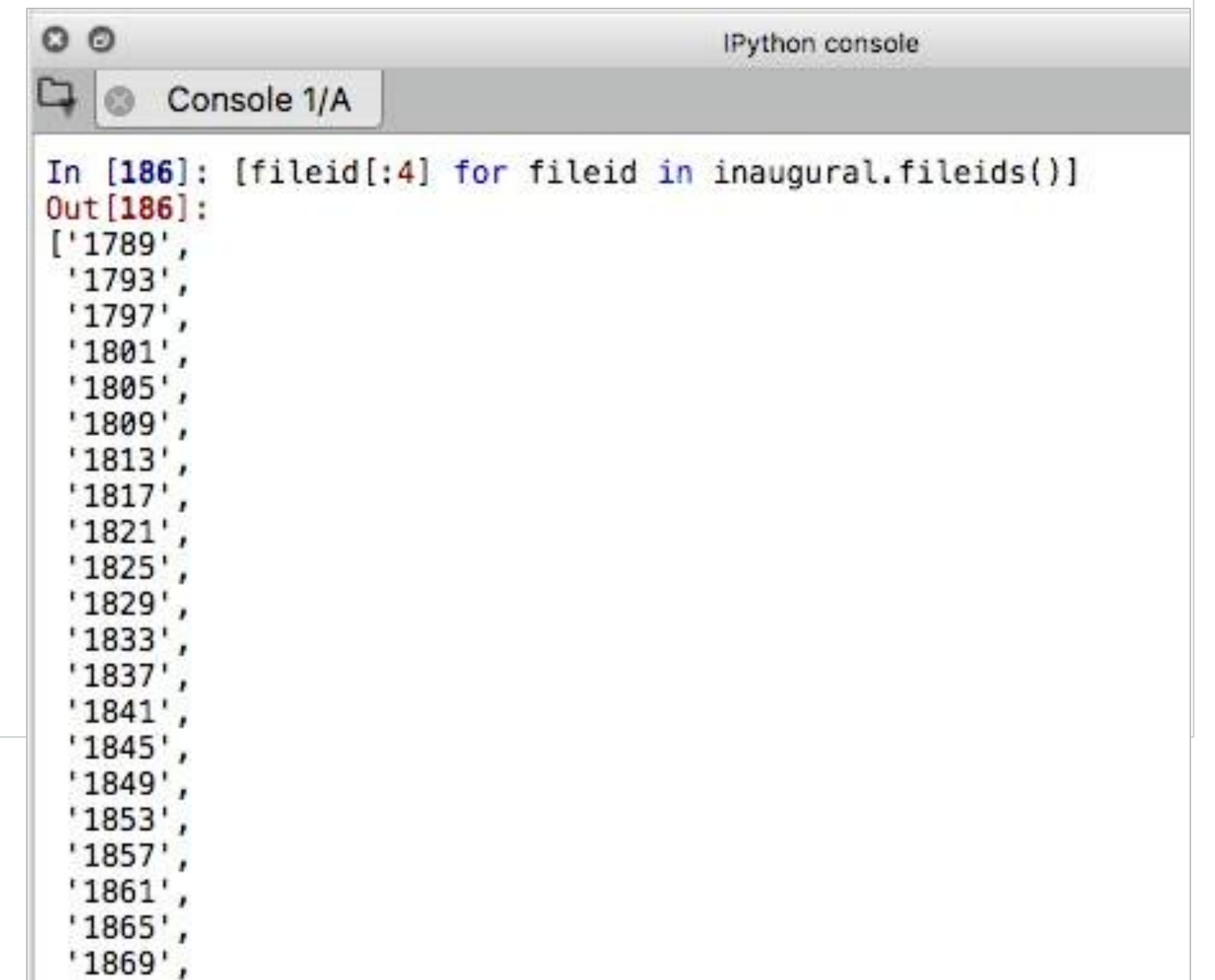
- This corpus has each presidential address over time, with a total of 55 addresses

```
# Import the Inaugural Address corpus
from nltk.corpus import inaugural

# Look at the file ids and categories
inaugural.fileids()

# We can extract the first four characters
# in each text to view the year for each
# address. We can pull it out with a for loop.
[fileid[:4] for fileid in inaugural.fileids()]
```

Script



```
IPython console
Console 1/A

In [186]: [fileid[:4] for fileid in inaugural.fileids()]
Out[186]:
['1789',
'1793',
'1797',
'1801',
'1805',
'1809',
'1813',
'1817',
'1821',
'1825',
'1829',
'1833',
'1837',
'1841',
'1845',
'1849',
'1853',
'1857',
'1861',
'1865',
'1869',
```



# The Inaugural Address corpus

---

- How have presidents used the word "America" and "citizen" over time?

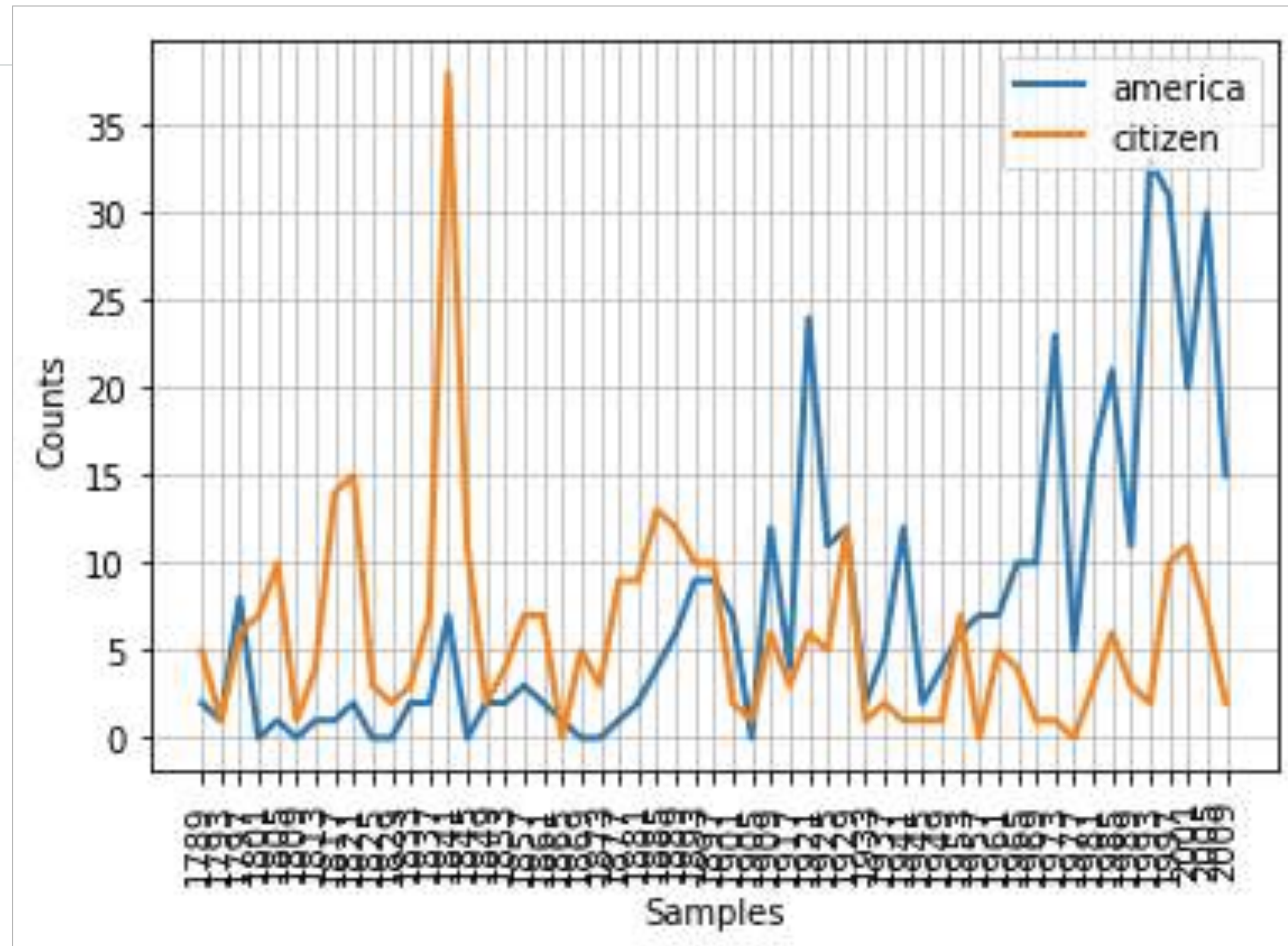
```
# Let's plot the word usage over each text - we're converting the  
# text to lowercase, then check if the target starts with 'america' or  
# 'citizen' with startswith() to include words like "American" and "citizens".  
cfd = nltk.ConditionalFreqDist(  
    (target, fileid[:4])  
    for fileid in inaugural.fileids()  
    for w in inaugural.words(fileid)  
    for target in ['america', 'citizen']  
    if w.lower().startswith(target))
```

Script

# The Inaugural Address corpus

```
# Let's plot the distribution  
cfd.plot()
```

Script





# Text mining in different languages

- NLTK comes with corpora in different languages, but keep in mind that you may need to manipulate character encodings in Python

```
# Make sure that the udhr and udhr2 corpora are downloaded  
nltk.download()
```

Script

Collections	Corpora	Models	All Packages
Identifier	Name	Size	Status
swadesh	Swadesh Wordlists	22.3 KB	installed
switchboard	Switchboard Corpus Sample	772.6 KB	not installed
timit	TIMIT Corpus Sample	21.2 MB	installed
toolbox	Toolbox Sample Files	244.7 KB	installed
treebank	Penn Treebank Sample	1.7 MB	installed
twitter_samples	Twitter Samples	15.3 MB	not installed
udhr	Universal Declaration of Human Rights Corpus	1.1 MB	installed
udhr2	Universal Declaration of Human Rights Corpus (Unicode Versi	1.6 MB	installed
unicode_samples	Unicode Samples	1.2 KB	installed
universal_treebanks_v20	Universal Treebanks Version 2.0	24.7 MB	not installed
verbnet	VerbNet Lexicon, Version 2.1	316.1 KB	not installed
webtext	Web Text Corpus	631.1 KB	installed
wordnet	WordNet	10.3 MB	installed
wordnet_ic	WordNet-InfoContent	11.5 MB	installed
words	Word Lists	740.0 KB	installed
ycoe	York-Toronto-Helsinki Parsed Corpus of Old English Prose	0.5 KB	not installed

Download Refresh

Server Index:

Download Directory:



# Text mining in different languages

- NLTK comes with corpora in different languages, but keep in mind that you may need to manipulate character encodings in Python

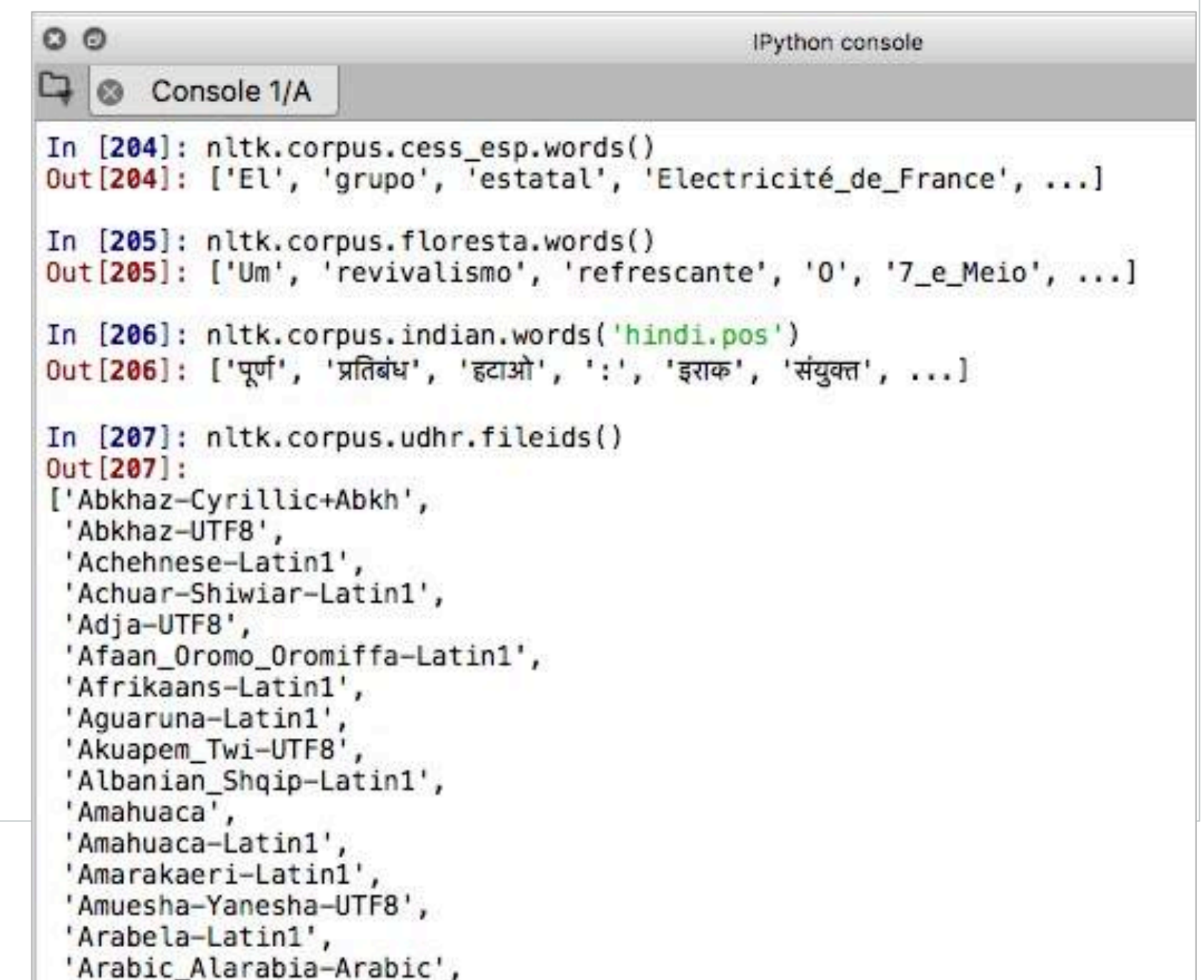
```
# Spanish  
nltk.corpus.cess_esp.words()
```

```
# Portuguese  
nltk.corpus.floresta.words()
```

```
# Hindi  
nltk.corpus.indian.words('hindi.pos')
```

```
# The udhr corpus contains the Universal Declaration  
# of Human Rights in over 300 languages  
nltk.corpus.udhr.fileids()
```

Script



The screenshot shows an IPython console window titled 'IPython console' with a sub-tab 'Console 1/A'. It displays the following code and output:

```
In [204]: nltk.corpus.cess_esp.words()  
Out[204]: ['El', 'grupo', 'estatal', 'Electricité_de_France', ...]  
  
In [205]: nltk.corpus.floresta.words()  
Out[205]: ['Um', 'revivalismo', 'refrescante', '0', '7_e_Meio', ...]  
  
In [206]: nltk.corpus.indian.words('hindi.pos')  
Out[206]: ['पूर्ण', 'प्रतिबंध', 'हटाओ', ':', 'इराक', 'संयुक्त', ...]  
  
In [207]: nltk.corpus.udhr.fileids()  
Out[207]: ['Abkhaz-Cyrillic+Abkh',  
          'Abkhaz-UTF8',  
          'Achehnese-Latin1',  
          'Achuar-Shiwiari-Latin1',  
          'Adja-UTF8',  
          'Afaan_Oromo_Oromiffa-Latin1',  
          'Afrikaans-Latin1',  
          'Aguaruna-Latin1',  
          'Akuapem-Twi-UTF8',  
          'Albanian_Shqip-Latin1',  
          'Amahuaca',  
          'Amahuaca-Latin1',  
          'Amarakaeri-Latin1',  
          'Amuesha-Yanesha-UTF8',  
          'Arabela-Latin1',  
          'Arabic_Alarabia-Arabic',
```

# Text mining in different languages

- Let's use a conditional frequency distribution to examine the differences in word lengths for a selection of languages

```
# Make sure udhr is imported  
from nltk.corpus import udhr
```

Script

```
# Define the languages you want to use  
languages = ['Chickasaw', 'English', 'German_Deutsch',  
             'Greenlandic_Inuktitut', 'Icelandic_Yslenska', 'Norwegian']
```

```
# Set the conditional frequency distribution for word length
```

```
cfd = nltk.ConditionalFreqDist(  
    (lang, len(word))  
    for lang in languages  
    for word in udhr.words(lang + '-Latin1'))
```

Specify that we want to use the languages with Latin characters

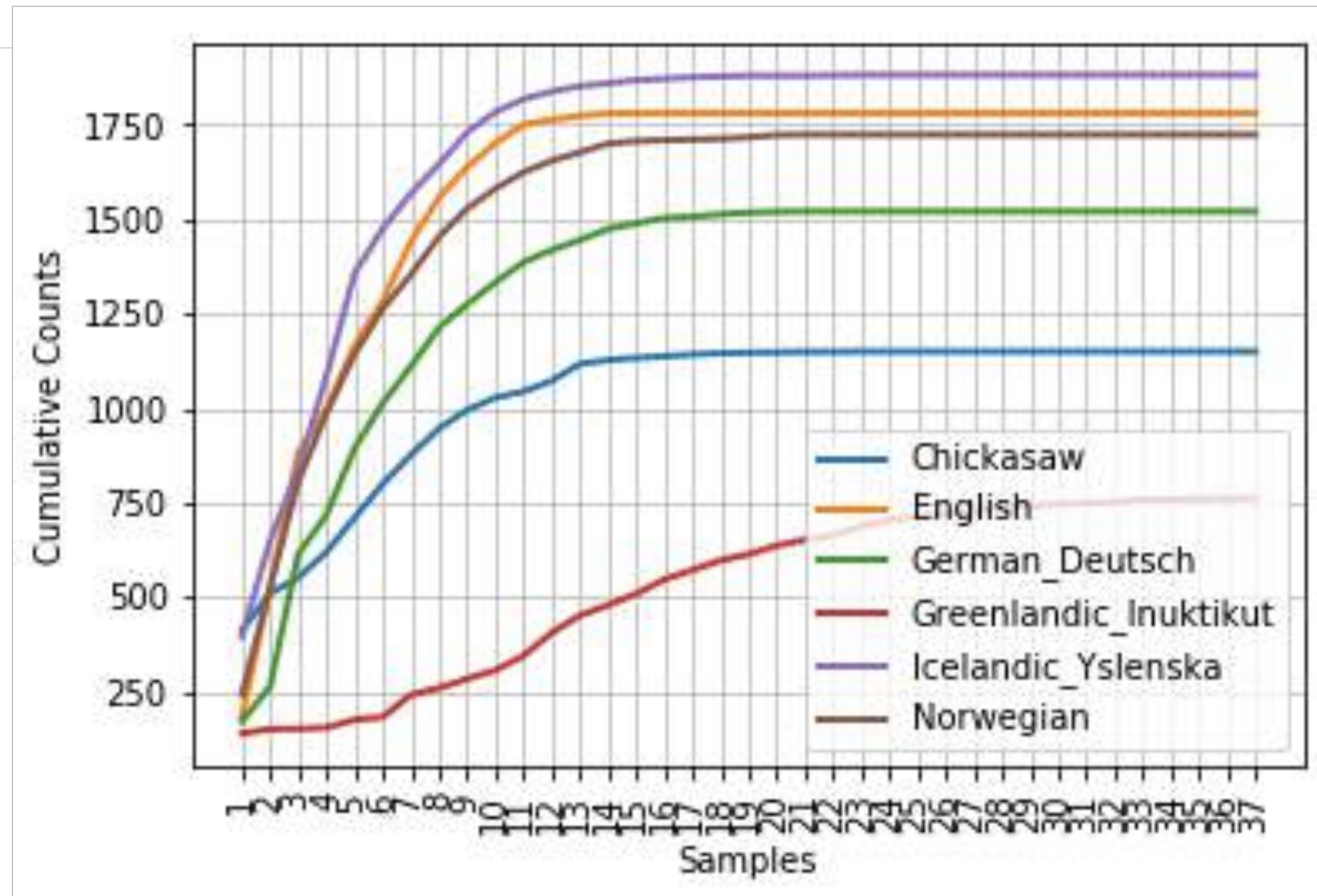




# Text mining in different languages

```
# Let's plot the distribution  
cfd.plot(cumulative = True)
```

Script



Unfortunately, for many languages, documentation is not available

# Basic functions for corpora

Example	Description
<code>fileids()</code>	the files of the corpus
<code>fileids([categories])</code>	the files of the corpus corresponding to these categories
<code>categories()</code>	the categories of the corpus
<code>categories([fileids])</code>	the categories of the corpus corresponding to these files
<code>raw()</code>	the raw content of the corpus
<code>raw(fileids=[f1,f2,f3])</code>	the raw content of the specified files
<code>raw(categories=[c1,c2])</code>	the raw content of the specified categories
<code>words()</code>	the words of the whole corpus
<code>words(fileids=[f1,f2,f3])</code>	the words of the specified fileids
<code>words(categories=[c1,c2])</code>	the words of the specified categories
<code>sents()</code>	the sentences of the whole corpus
<code>sents(fileids=[f1,f2,f3])</code>	the sentences of the specified fileids
<code>sents(categories=[c1,c2])</code>	the sentences of the specified categories



# Loading your own corpus

- In order to load your own text, you have to import PlaintextCorpusReader

```
# Import the PlaintextCorpusReader  
from nltk.corpus import PlaintextCorpusReader
```

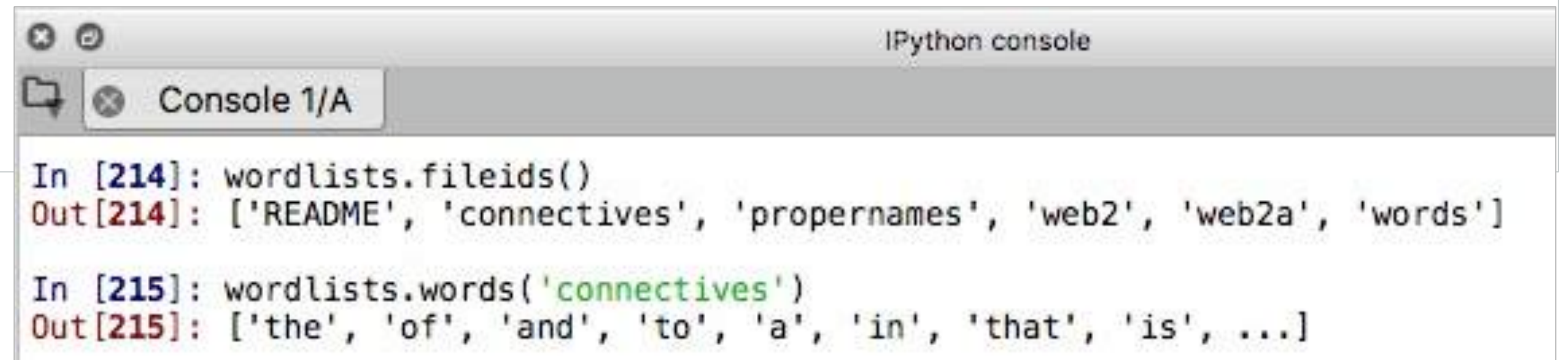
```
# Whatever the file location, set it to 'corpus_root'  
corpus_root = '/usr/share/dict'
```

```
# The second parameter can be a list of file ids or patterns that match  
# all file ids
```

```
wordlists = PlaintextCorpusReader(corpus_root, '.*')  
wordlists.fileids()
```

```
wordlists.words('connectives')
```

Script



IPython console

Console 1/A

```
In [214]: wordlists.fileids()  
Out[214]: ['README', 'connectives', 'propernames', 'web2', 'web2a', 'words']  
  
In [215]: wordlists.words('connectives')  
Out[215]: ['the', 'of', 'and', 'to', 'a', 'in', 'that', 'is', ...]
```

# Overview

---

1. Introduction to text mining
2. Searching through text
3. Identifying characteristics of text
4. Identifying common word pairs
5. Using conditionals
6. Mapping word distributions
7. Conditional frequency distributions



# Conditional frequency distributions

---

- Frequency distributions compute the number of occurrences in each category so we can study the differences between categories
- A **conditional frequency distribution** is a collection of frequency distributions, each one for a different "condition"
- **FreqDist()** takes a simple list as an input, but **ConditionalFreqDist()** takes a list of pairs (i.e. a pair could consist of a category and a term)

# Counting words by category

---

- We will break this down in the next few slides

```
# Here is how we have used ConditionalFreqDist() in the past:  
from nltk.corpus import brown  
  
# Calculate the word frequency by genre  
cfd = nltk.ConditionalFreqDist(  
    (genre, word)  
    for genre in brown.categories()  
    for word in brown.words(categories = genre))
```

Script

# Counting words by category

Script

```
# Let's take two genres, news and romance, and loop over every word to  
# produce pairs of the genre and the word:
```

```
genre_word = [(genre, word)  
               for genre in ['news', 'romance']  
               for word in brown.words(categories = genre)]
```

```
len(genre_word)  # 170576
```

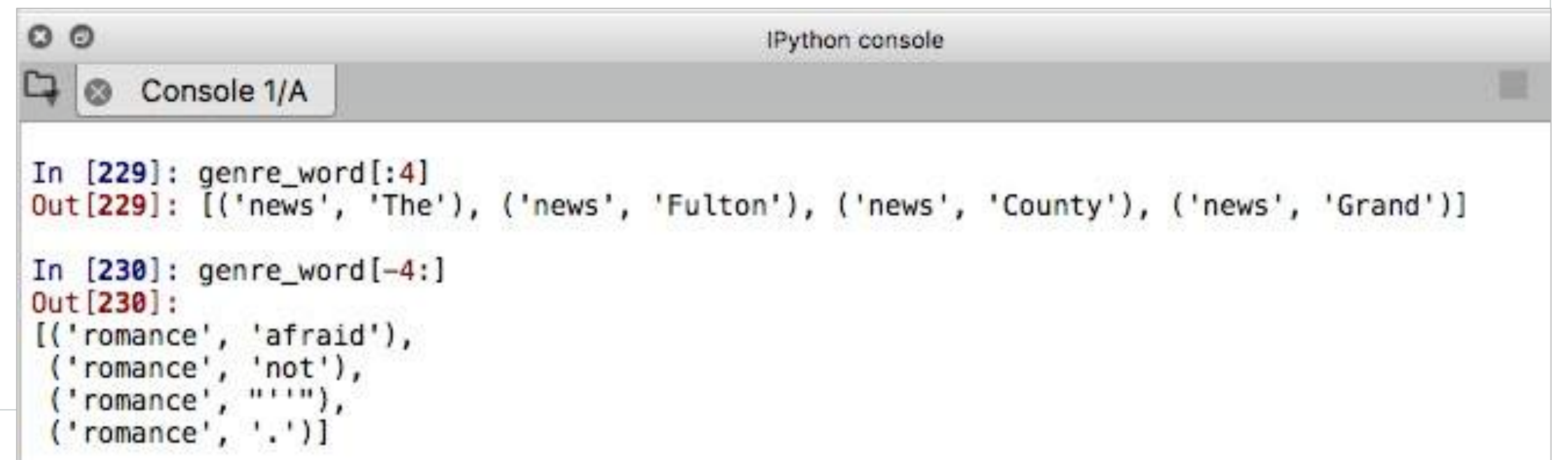
```
# The first pairs will have the genre 'news', and then the last pairs will  
# have the genre 'romance' because that's the order we put the genres in.
```

```
# The first four pairs
```

```
genre_word[:4]
```

```
# The last four pairs
```

```
genre_word[-4:]
```



```
IPython console  
Console 1/A  
In [229]: genre_word[:4]  
Out[229]: [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]  
  
In [230]: genre_word[-4:]  
Out[230]: [('romance', 'afraid'),  
            ('romance', 'not'),  
            ('romance', ''),  
            ('romance', '.')] 
```



# Counting words by category

*# Now we can use ConditionalFreqDist() on the list of pairs we created*

```
cfd = nltk.ConditionalFreqDist(genre_word)
```

*# We can type in the variable to inspect it and check the conditions*

```
cfd
```

```
cfd.conditions()
```

*# We can also check the frequency*

*# distributions for the genres*

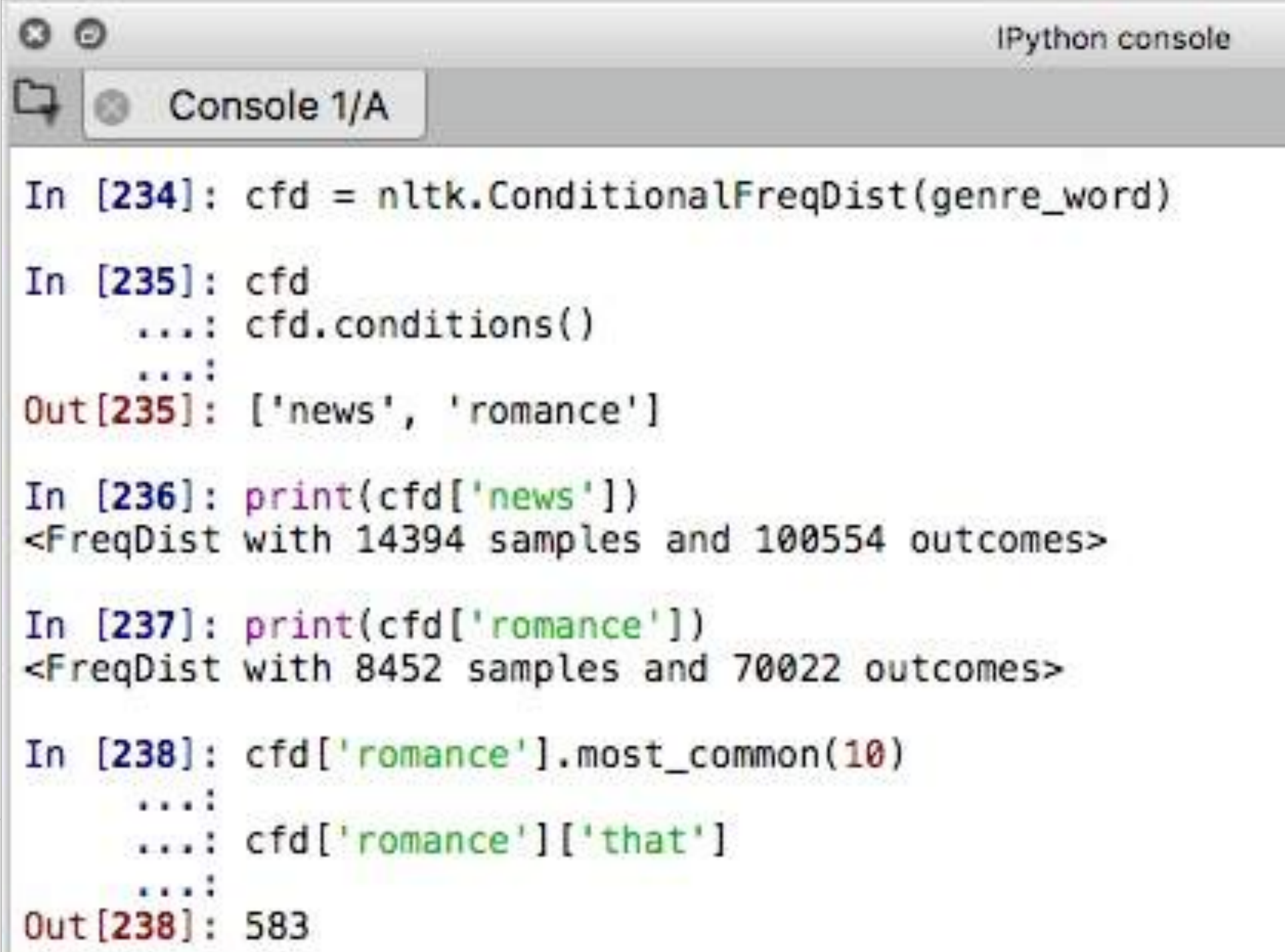
```
print(cfd['news'])
```

```
print(cfd['romance'])
```

```
cfd['romance'].most_common(10)
```

```
cfd['romance']['that']
```

Script



IPython console

Console 1/A

```
In [234]: cfd = nltk.ConditionalFreqDist(genre_word)

In [235]: cfd
...: cfd.conditions()
...:
Out[235]: ['news', 'romance']

In [236]: print(cfd['news'])
<FreqDist with 14394 samples and 100554 outcomes>

In [237]: print(cfd['romance'])
<FreqDist with 8452 samples and 70022 outcomes>

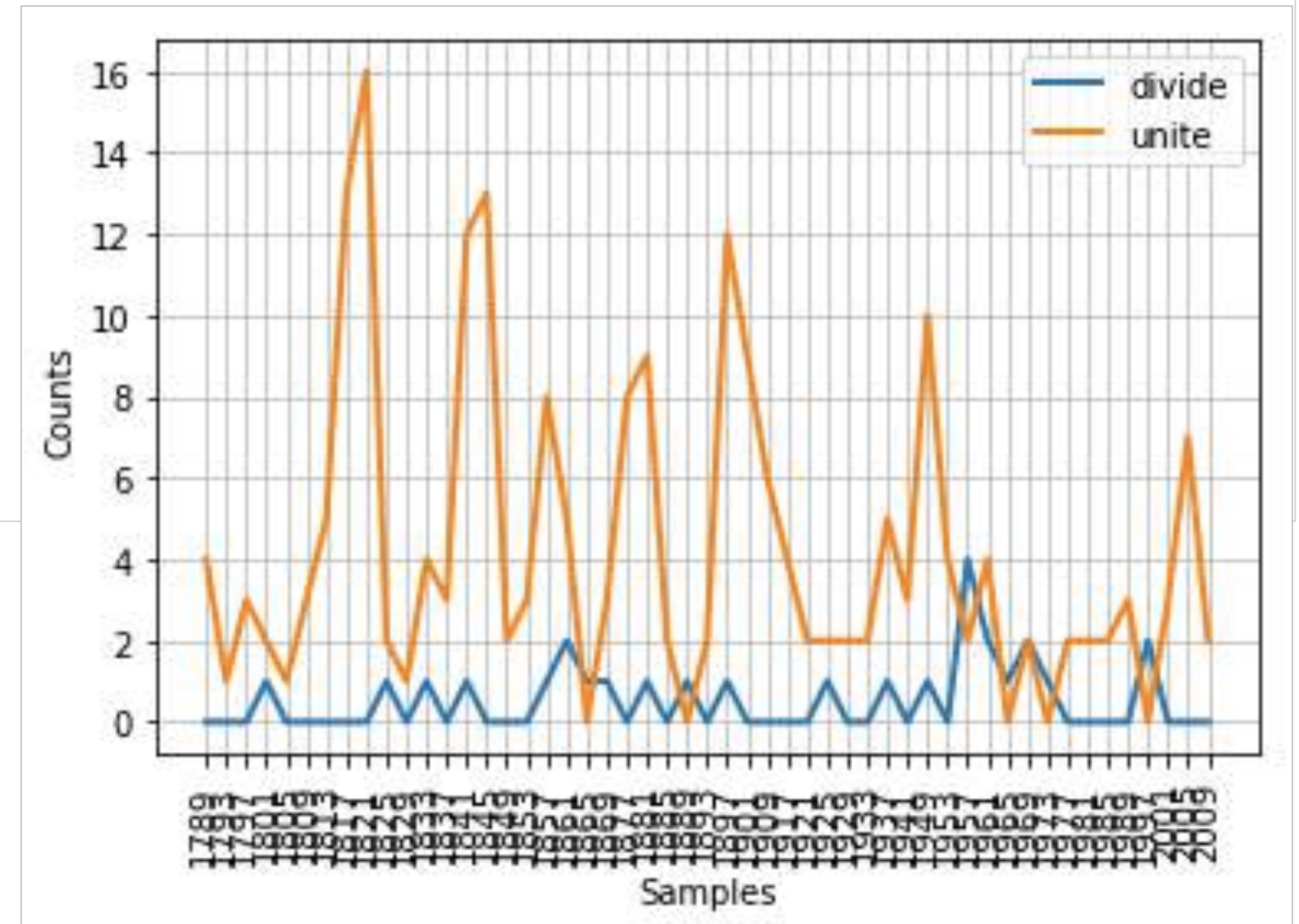
In [238]: cfd['romance'].most_common(10)
...:
...: cfd['romance']['that']
...:
Out[238]: 583
```

# Plotting distribution

*# Here, we're combining the previous code into one block. This is  
# similar to the Inaugural Address plot. We create the conditional frequency  
# distribution and counted the occurrences for each word.*

Script

```
cfd = nltk.ConditionalFreqDist(  
    (target, fileid[:4])  
    for fileid in inaugural.fileids()  
    for w in inaugural.words(fileid)  
    for target in ['unite', 'divide']  
    if w.lower().startswith(target))  
  
cfd.plot()
```





# Tabulating occurrences

---

- For `plot()` and `tabulate()`, we can specify which conditions to display with `conditions = parameter`.
- We can limit the samples to display with a `samples = parameter`.

```
from nltk.corpus import udhr

languages = ['Chickasaw', 'English', 'German_Deutsch',
            'Greenlandic_Inuktitut', 'Icelandic_Yslenska', 'Norwegian']

# Set the conditional frequency distribution for word length
cfd = nltk.ConditionalFreqDist(
    (lang, len(word))
    for lang in languages
    for word in udhr.words(lang + '-Latin1'))
```

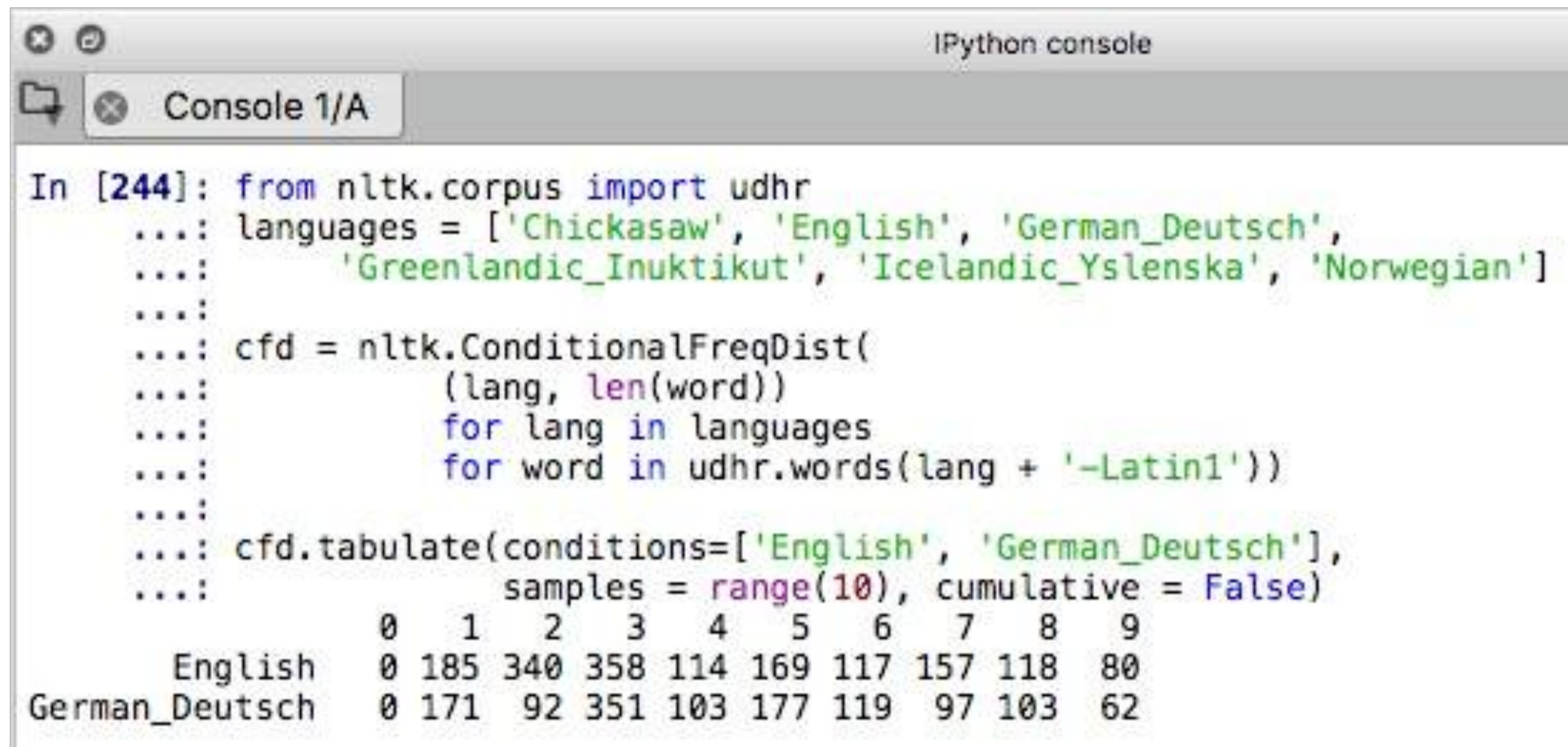
Script



# Tabulating occurrences

```
cfid.tabulate(conditions = ['English', 'German_Deutsch'],  
              samples = range(10), cumulative = False)
```

Script



```
IPython console  
Console 1/A  
In [244]: from nltk.corpus import udhr  
...: languages = ['Chickasaw', 'English', 'German_Deutsch',  
...:              'Greenlandic_Inuktitut', 'Icelandic_Yslenska', 'Norwegian']  
...:  
...: cfd = nltk.ConditionalFreqDist(  
...:     (lang, len(word))  
...:     for lang in languages  
...:     for word in udhr.words(lang + '-Latin1'))  
...:  
...: cfd.tabulate(conditions=['English', 'German_Deutsch'],  
...:              samples = range(10), cumulative = False)  
...:  
      English      0  1  2  3  4  5  6  7  8  9  
German_Deutsch  0 171  92 351 103 177 119  97 103  62
```

*Now we can load a large quantity of data into a conditional frequency distribution, and explore it by plotting or tabulating selected conditions and samples.*