

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. Create your own tokens
2. Automate parts-of-speech taggers
3. Define your own dictionaries

Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Categorizing and tagging words

- **Word classes** (also known as **lexical categories**) are ways for us categorize words for language processing tasks
- The process of classifying words into their parts of speech (nouns, verbs, adjectives, etc) and labeling them accordingly is known as **part-of-speech tagging** or **POS-tagging**
- We can do a simple analysis to automatically tag words

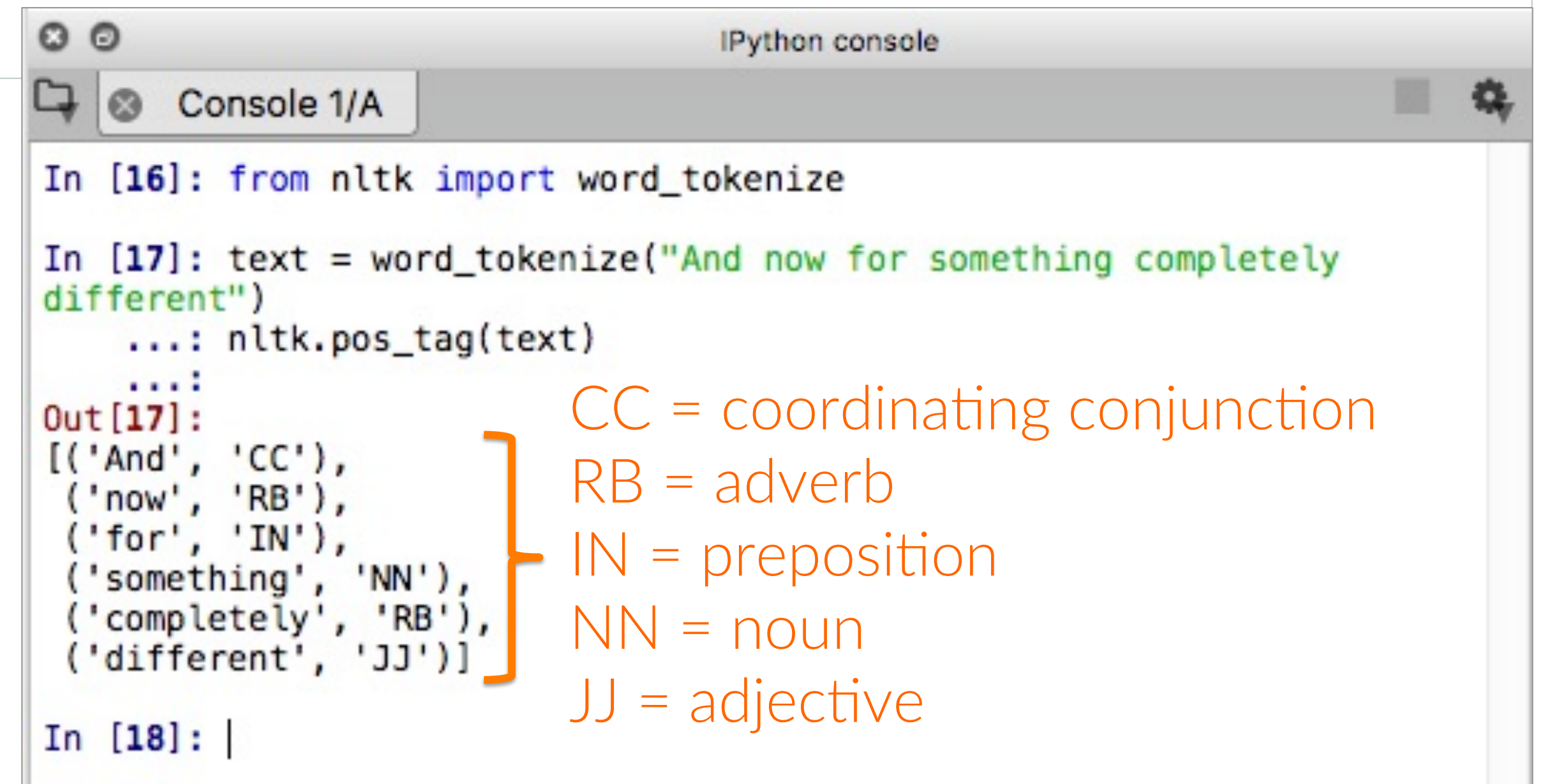
Using a tagger

```
# Import nltk and word_tokenize
import nltk
from nltk import word_tokenize
```

Script

```
# What are the parts of speech of the sentence below?
text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

If you want to look at the entire list of categorizations, you can type in `nltk.help.upenn_tagset()` to see the full list



```
IPython console
Console 1/A

In [16]: from nltk import word_tokenize

In [17]: text = word_tokenize("And now for something completely
different")
...: nltk.pos_tag(text)
...:
Out[17]:
[('And', 'CC'),
 ('now', 'RB'),
 ('for', 'IN'),
 ('something', 'NN'),
 ('completely', 'RB'),
 ('different', 'JJ')]

In [18]: |
```

CC = coordinating conjunction
RB = adverb
IN = preposition
NN = noun
JJ = adjective

Using a tagger

```
# What if we use homonyms (same word with a different meaning)?
text = word_tokenize("They refuse to permit us to obtain the refuse permit")
nltk.pos_tag(text)
```

Script

The POS tagger was able to distinguish between the different meanings of "refuse" and "permit"!

```
IPython console
Console 1/A

In [19]: text = word_tokenize("They refuse to permit us to obtain the refuse permit")
...: nltk.pos_tag(text)
...:
Out[19]:
[('They', 'PRP'),
 ('refuse', 'VBP'),
 ('to', 'TO'),
 ('permit', 'VB'),
 ('us', 'PRP'),
 ('to', 'TO'),
 ('obtain', 'VB'),
 ('the', 'DT'),
 ('refuse', 'NN'),
 ('permit', 'NN')]

[('They', 'PRP'),
 ('refuse', 'VBP'),
 ('to', 'TO'),
 ('permit', 'VB'),
 ('us', 'PRP'),
 ('to', 'TO'),
 ('obtain', 'VB'),
 ('the', 'DT'),
 ('refuse', 'NN'),
 ('permit', 'NN')]
```


Using a tagger

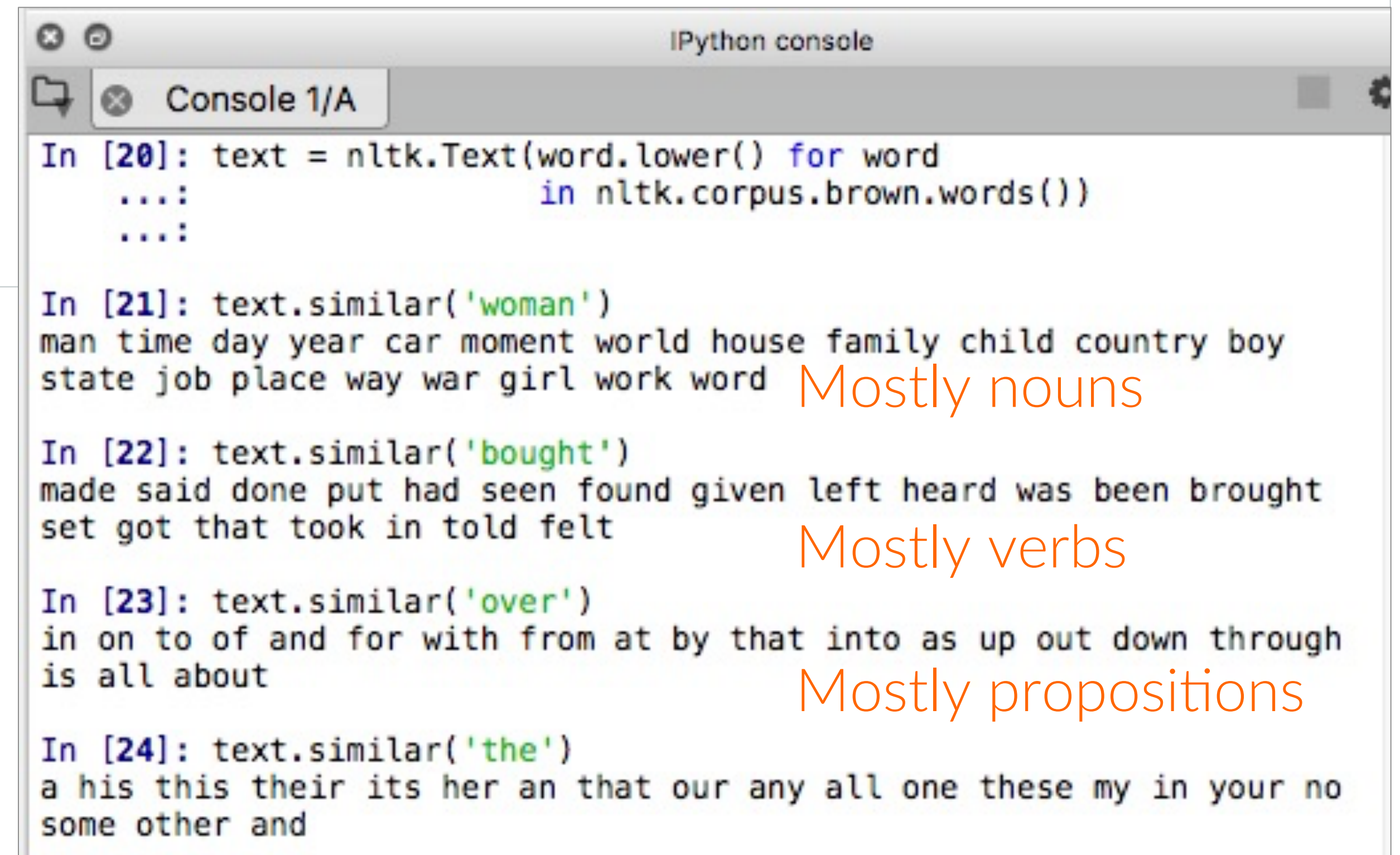
```
# How can we identify words that are used in similar context?
# First, we need to create the word list.
text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())

text.similar('woman')
text.similar('bought')
text.similar('over')
text.similar('the')
```

Script

The `text.similar` method takes the word we input, finds all the contexts of that word, and then finds words that appear in the same context

Parts of speech can help us understand the word distribution for analysis



```
IPython console
Console 1/A

In [20]: text = nltk.Text(word.lower() for word
...:           in nltk.corpus.brown.words())
...:

In [21]: text.similar('woman')
man time day year car moment world house family child country boy
state job place way war girl work word

In [22]: text.similar('bought')
made said done put had seen found given left heard was been brought
set got that took in told felt

In [23]: text.similar('over')
in on to of and for with from at by that into as up out down through
is all about

In [24]: text.similar('the')
a his this their its her an that our any all one these my in your no
some other and
```

Mostly nouns

Mostly verbs

Mostly propositions

Tagged tokens

- A **tagged token** is represented by a **tuple** (a sequence of elements) where the first element is the word, and the second element is the tag



```
# We can create one of these special tuples from the standard string
# representation of a tagged token, using the function str2tuple():
tagged_token = nltk.tag.str2tuple('fly/NN')
tagged_token
tagged_token[0]
tagged_token[1]
```

Script

The screenshot shows an IPython console window titled "IPython console" with a sub-tab "Console 1/A". It displays the following interactions:

```
In [25]: tagged_token = nltk.tag.str2tuple('fly/NN')
...: tagged_token
...:
Out[25]: ('fly', 'NN')

In [26]: tagged_token[0]
Out[26]: 'fly'

In [27]: tagged_token[1]
Out[27]: 'NN'
```


Creating tagged tokens

We can create a list of tokens manually

Script

```
sent = ''' The/AT grand/JJ jury/NN commented/VBD on/IN a/AT number/NN of/IN  
other/AP topics/NNS ,/, AMONG/IN them/PPO the/AT Atlanta/NP and/CC Fulton/NP-tl  
County/NN-tl purchasing/VBG departments/NNS which/WDT it/PPS said/VBD ``/`` ARE/  
BER well/QL operated/VBN and/CC follow/VB generally/RB accepted/VBN practices/  
NNS which/WDT inure/VB to/IN the/AT best/JJT interest/NN of/IN both/ABX  
governments/NNS ''/'' ./.
```

```
[nltk.tag.str2tuple(t) for t in sent.split()]
```



```
IPython console
Console 1/A
In [29]: [nltk.tag.str2tuple(t) for t in sent.split()]
Out[29]:
[('The', 'AT'),
 ('grand', 'JJ'),
 ('jury', 'NN'),
 ('commented', 'VBD'),
 ('on', 'IN'),
 ('a', 'AT'),
 ('number', 'NN'),
 ('of', 'IN'),
 ('other', 'AP'),
 ('topics', 'NNS'),
 (',', ','),
 ('AMONG', 'IN'),
 ('them', 'PPO'),
 ('the', 'AT'),
 ('Atlanta', 'NP'),
 ('and', 'CC'),
 ('Fulton', 'NP-tl'),
 ('County', 'NN-tl'),
 ('purchasing', 'VBG'),
 ('departments', 'NNS'),
 ('which', 'WDT'),
 ('it', 'PPS'),
 ('said', 'VBD'),
 ('', ''),
 ('ARE', 'BER'),
 ('well', 'QL'),
 ('operated', 'VBN'),
 ('and', 'CC'),
 ('follow', 'VB'),
 ('generally', 'RB'),
 ('accepted', 'VBN'),
 ('practices', 'NNS'),
 ('which', 'WDT'),
 ('inure', 'VB'),
 ('to', 'IN'),
 ('the', 'AT'),
 ('best', 'JJT'),
 ('interest', 'NN'),
 ('of', 'IN'),
 ('both', 'ABX'),
 ('governments', 'NNS'),
 ('', ''),
 ('./.', '.')]

And here is our list
of tagged tokens!
```

Built-in taggers

- Several corpora in NLTK have been tagged already

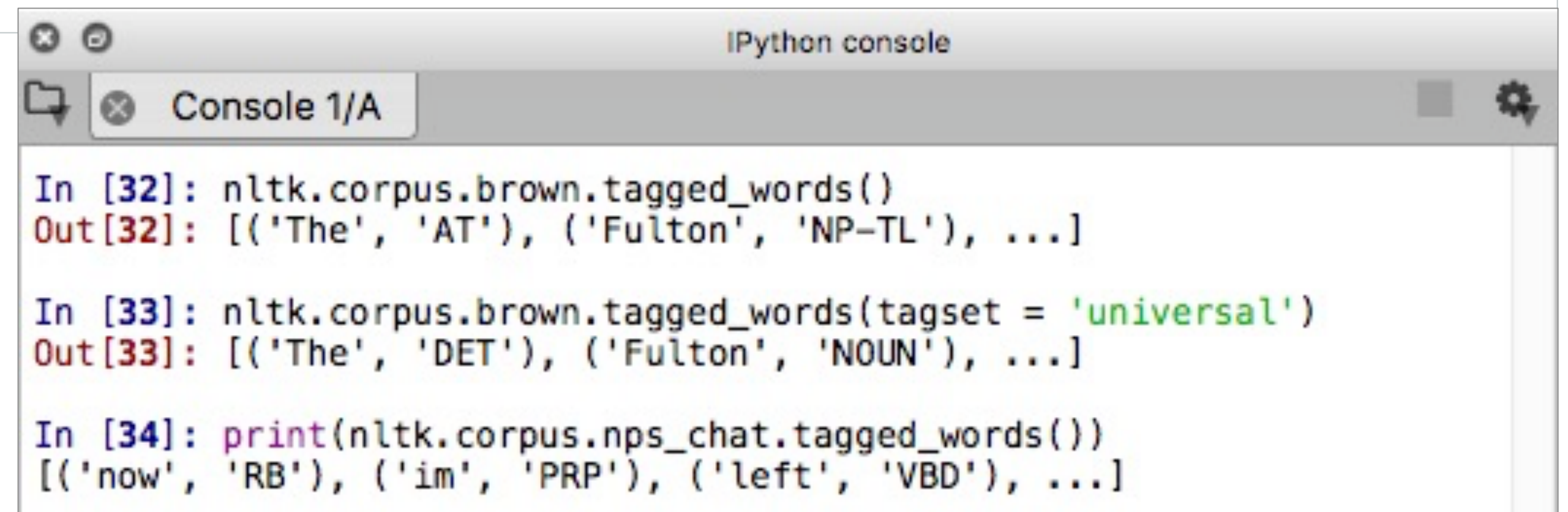
```
# We can use the tagged_words() syntax to view the tags of the words.  
nltk.corpus.brown.tagged_words()
```

Script

```
# The universal tagset contains different terms for parts of speech.  
nltk.corpus.brown.tagged_words(tagset = 'universal')
```

```
# If a corpus has tagged text, you'll be able to use the tagged_words() method.  
print(nltk.corpus.nps_chat.tagged_words())
```

You can learn more about the universal tagset here:
<https://github.com/slavpetrov/universal-pos-tags>

An IPython console window titled "IPython console" with a sub-tab "Console 1/A". It displays three code snippets and their corresponding outputs. The first snippet calls `nltk.corpus.brown.tagged_words()` and the output is a list of tuples like `(('The', 'AT'), ('Fulton', 'NP-TL'), ...)`. The second snippet calls `nltk.corpus.brown.tagged_words(tagset = 'universal')` and the output is a list of tuples like `(('The', 'DET'), ('Fulton', 'NOUN'), ...)`. The third snippet calls `print(nltk.corpus.nps_chat.tagged_words())` and the output is a list of tuples like `(('now', 'RB'), ('im', 'PRP'), ('left', 'VBD'), ...)`.

```
IPython console  
Console 1/A  
In [32]: nltk.corpus.brown.tagged_words()  
Out[32]: (('The', 'AT'), ('Fulton', 'NP-TL'), ...]  
  
In [33]: nltk.corpus.brown.tagged_words(tagset = 'universal')  
Out[33]: (('The', 'DET'), ('Fulton', 'NOUN'), ...]  
  
In [34]: print(nltk.corpus.nps_chat.tagged_words())  
[('now', 'RB'), ('im', 'PRP'), ('left', 'VBD'), ...]
```

Universal POS tags

| Tag | Meaning | English example |
|------|---------------------|---|
| ADJ | adjective | <i>new, good, high, special, big, local</i> |
| ADP | adposition | <i>on, of, at, with, by, into, under</i> |
| ADV | adverb | <i>really, already, still, early, now</i> |
| CONJ | conjunction | <i>and, or, but, if, while, although</i> |
| DET | determiner, article | <i>the, a, some, most, every, no, which</i> |
| NOUN | noun | <i>year, home, costs, time, Africa</i> |
| NUM | numeral | <i>twenty-four, fourth, 1991, 14:24</i> |
| PRT | particle | <i>at, on, out, over per, that, up, with</i> |
| PRON | pronoun | <i>he, their, her, its, my, I, us</i> |
| VERB | verb | <i>is, say, told, given, playing, would</i> |
| . | punctuation marks | <i>., ; !</i> |
| X | other | <i>ersatz, esprit, dunno, gr8, univeristy</i> |

Tabulating tags

- Let's see which of these tags are most common in the news

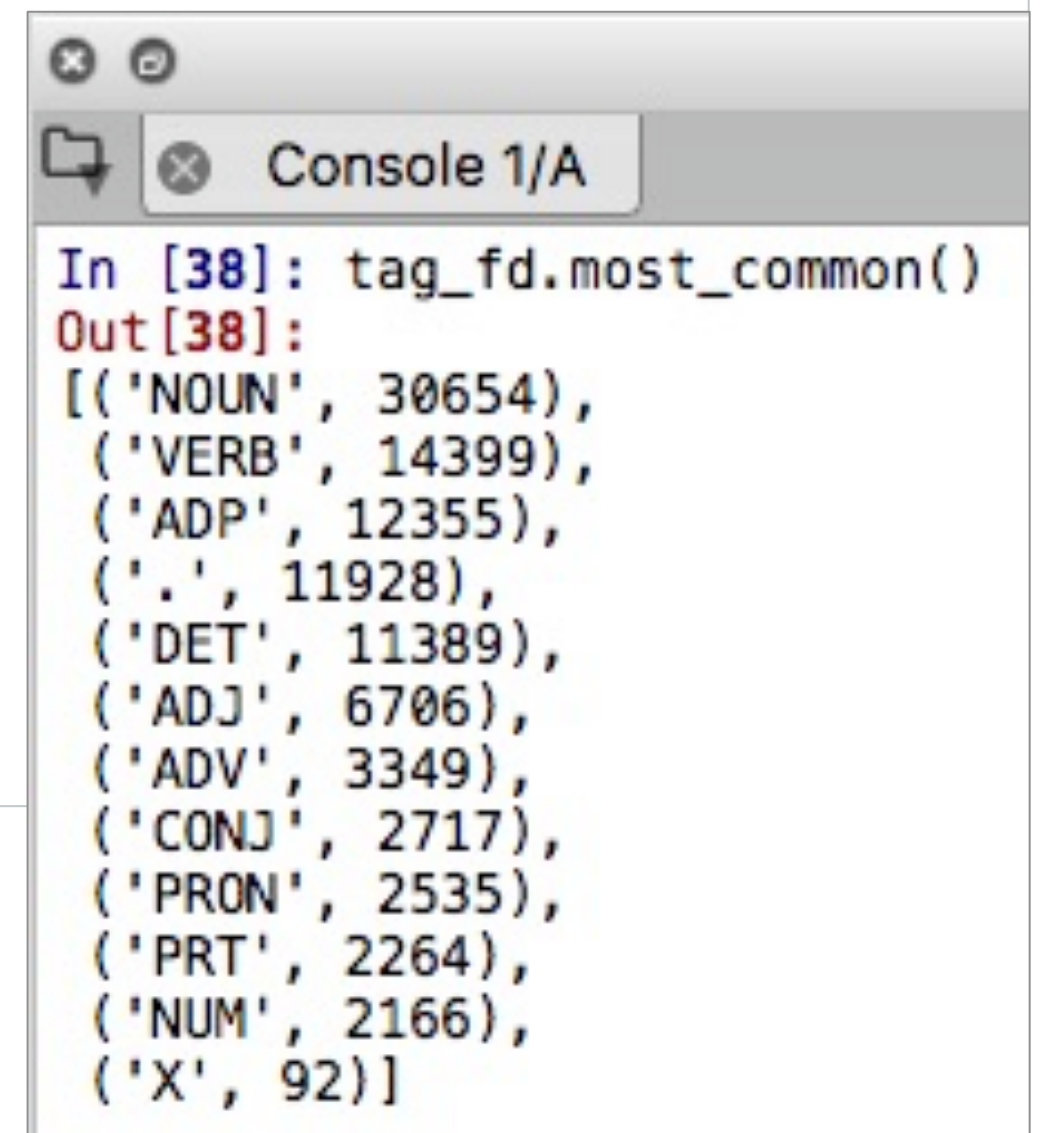
```
# Make sure to import the Brown corpus first.
from nltk.corpus import brown

# Now we'll pull all the words in the "news" category with a universal tag.
brown_news_tagged = brown.tagged_words(categories = 'news', tagset = 'universal')

# We'll use the FreqDist() syntax from nltk to calculate
# the frequency distribution of the tags.
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)

# What are the most common parts of speech?
tag_fd.most_common()
```

Script



Console 1/A

```
In [38]: tag_fd.most_common()
Out[38]:
[('NOUN', 30654),
 ('VERB', 14399),
 ('ADP', 12355),
 ('.', 11928),
 ('DET', 11389),
 ('ADJ', 6706),
 ('ADV', 3349),
 ('CONJ', 2717),
 ('PRON', 2535),
 ('PRT', 2264),
 ('NUM', 2166),
 ('X', 92)]
```

Exercise time!



Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Building and using taggers

Syntactic patterns: nouns

- What parts of speech typically appear before a noun?

```
# Let's create bigrams with the bigrams() syntax from NLTK.
```

```
word_tag_pairs = nltk.bigrams(brown_news_tagged)
```

```
# Now we will identify the words that appear before a noun in the bigrams  
# we just created.
```

```
noun_preceders = [a[1] for (a, b) in word_tag_pairs if b[1] == 'NOUN']
```

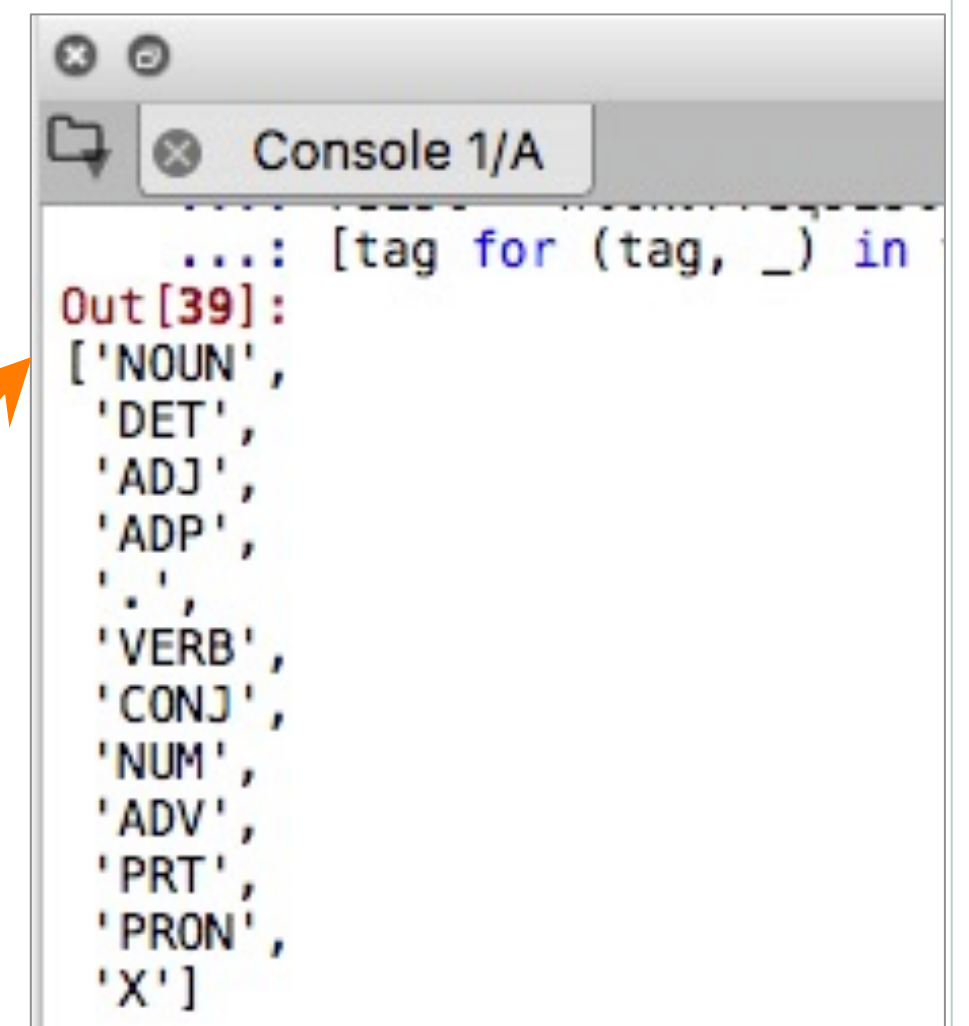
```
# We'll use the FreqDist() method on the word list we  
# just created to see which words occur most often before  
fdist = nltk.FreqDist(noun_preceders)
```

```
# Let's pull out only the part of speech term  
# (not the frequency).
```

```
[tag for (tag, _) in fdist.most_common()]
```

Looks like nouns and
determiners are the top two
parts of speech before nouns!

Script



```
...: [tag for (tag, _) in  
Out[39]:  
['NOUN',  
'DET',  
'ADJ',  
'ADP',  
'.',  
'VERB',  
'CONJ',  
'NUM',  
'ADV',  
'PRT',  
'PRON',  
'X']
```

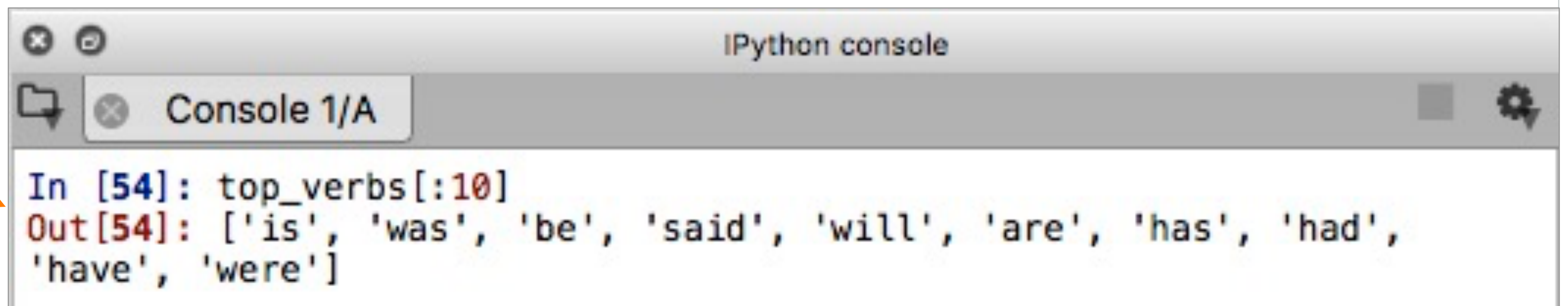
Syntactic patterns: verbs

- What are the most common verbs in news?

```
# Let's create bigrams with the bigrams() syntax from NLTK.  
word_tag_fd = nltk.FreqDist(brown_news_tagged)  
  
# Now we will identify the most common words that are tagged as verbs in the  
# frequency distribution.  
top_verbs = [wt[0] for (wt, _) in word_tag_fd.most_common() if wt[1] == 'VERB']  
  
# We can pull the top terms from the 'top_verbs' variable.  
top_verbs[:10]
```

Script

Here are the top verbs in news



The image shows a screenshot of an IPython console window. The window has a title bar that says "IPython console". Below the title bar, there is a tab labeled "Console 1/A". The console displays the following text:

```
In [54]: top_verbs[:10]  
Out[54]: ['is', 'was', 'be', 'said', 'will', 'are', 'has', 'had',  
'have', 'were']
```

An orange arrow points from the text "Here are the top verbs in news" to the output of the console.

Conditional frequency distribution

- Since words and tags are paired, we can treat the word as a condition and the tag as an event

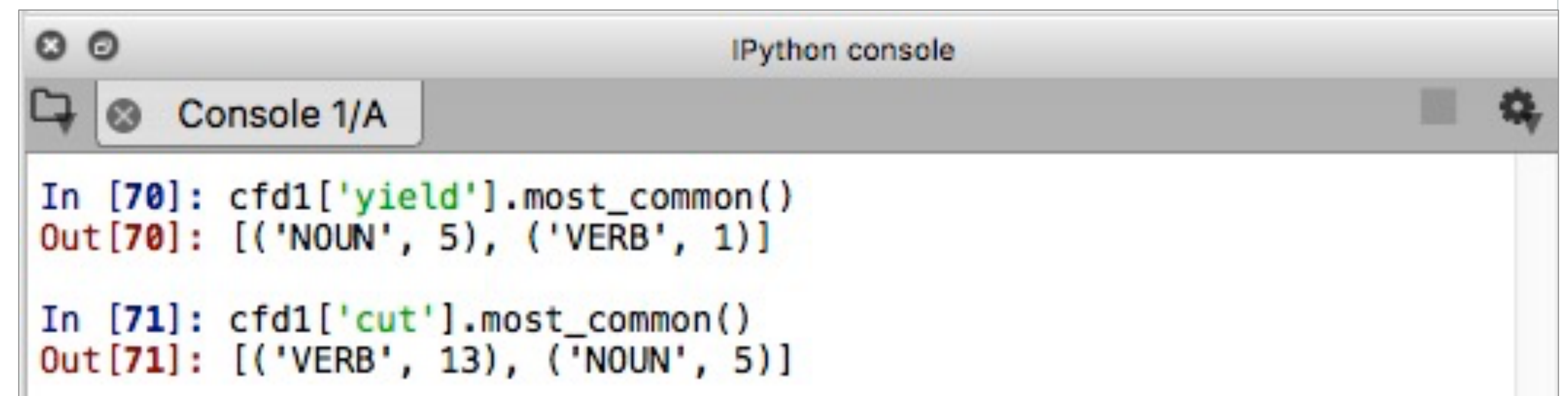
```
# First, we'll set up the Conditional Frequency Distribution  
cfd1 = nltk.ConditionalFreqDist(brown_news_tagged)
```

Script

```
# Let's look at some words that have multiple meanings to see how they're most  
# commonly used.  
cfd1['yield'].most_common()
```

```
cfd1['cut'].most_common()
```

Remember, Conditional Frequency Distribution shows the distribution across multiple variables – in this case, it's tallying up the word across the different parts of speech



```
IPython console  
Console 1/A  
In [70]: cfd1['yield'].most_common()  
Out[70]: [('NOUN', 5), ('VERB', 1)]  
  
In [71]: cfd1['cut'].most_common()  
Out[71]: [('VERB', 13), ('NOUN', 5)]
```


Conditional frequency distribution

- We can also reverse it and set the tags as the conditions and the word as the event so we can search by part of speech.

```
# First, we'll set up the Conditional Frequency Distribution  
brown_news_tagged_1 = brown.tagged_words(categories = 'news')
```

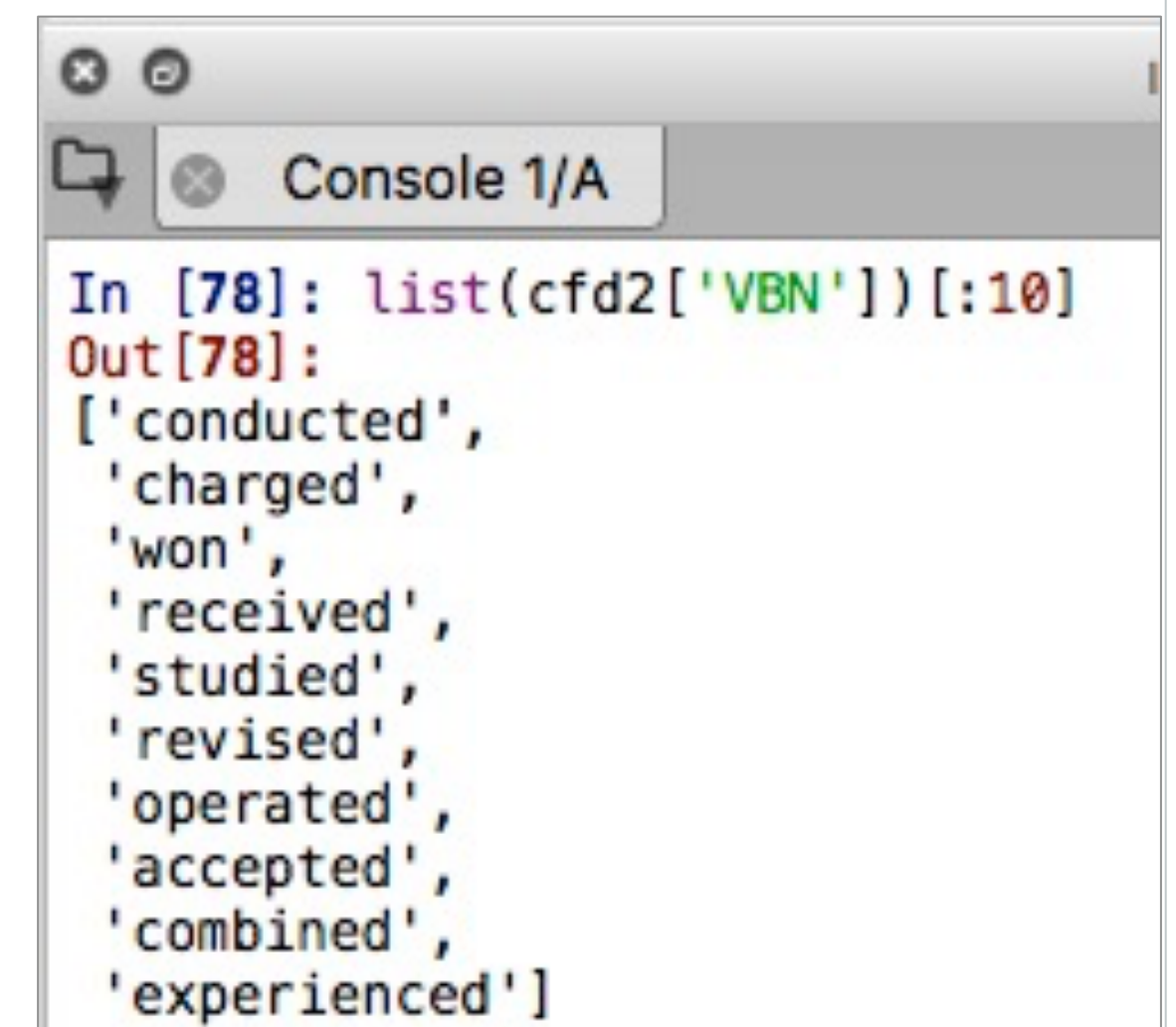
Script

```
cfd2 = nltk.ConditionalFreqDist((tag, word) for (word, tag)  
                                in brown_news_tagged_1)
```



Here we're switching the tags and the words so we can search by the tag

```
# We can pull the top ten terms that  
# are tagged as "verb past participle"  
list(cfd2['VBN'])[:10]
```



```
In [78]: list(cfd2['VBN'])[:10]  
Out[78]:  
['conducted',  
'charged',  
'won',  
'received',  
'studied',  
'revised',  
'operated',  
'accepted',  
'combined',  
'experienced']
```

Conditional frequency distribution

- To clarify the distinction between VBD (past tense) and VBN (past participle), let's find words which can be both VBD and VBN, and see surrounding text

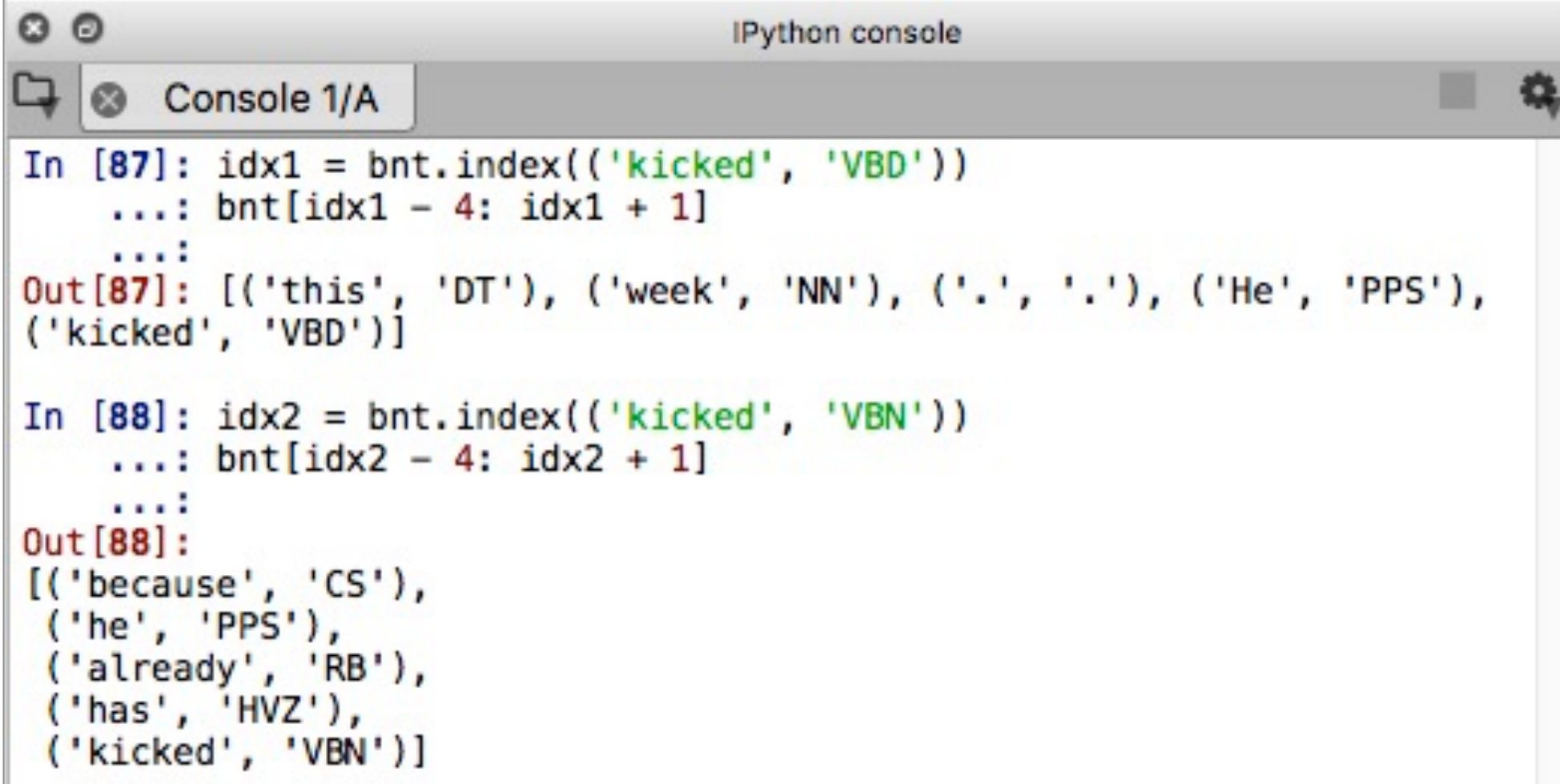
```
# First, we'll create the list where verbs are labeled as both VBD and VBN  
[w for w in cfd1.conditions() if 'VBD' in cfd1[w] and 'VBN' in cfd1[w]]
```

Script

```
# Let's shorten the variable to 'bnt' to make it easier to work with.  
bnt = brown_news_tagged_1
```

```
# Now, we can look at 'kicked' labeled as VBD and  
# pull out the indexed words before and after it  
# to see the words and parts of speech around it.  
idx1 = bnt.index(('kicked', 'VBD'))  
bnt[idx1 - 4: idx1 + 1]
```

```
# Let's see if the words are different for VBN.  
idx2 = bnt.index(('kicked', 'VBN'))  
bnt[idx2 - 4: idx2 + 1]
```



```
IPython console  
Console 1/A  
In [87]: idx1 = bnt.index(('kicked', 'VBD'))  
...: bnt[idx1 - 4: idx1 + 1]  
...:  
Out[87]: [('this', 'DT'), ('week', 'NN'), ('.', '.'), ('He', 'PPS'),  
( 'kicked', 'VBD')]  
  
In [88]: idx2 = bnt.index(('kicked', 'VBN'))  
...: bnt[idx2 - 4: idx2 + 1]  
...:  
Out[88]: [('because', 'CS'),  
( 'he', 'PPS'),  
( 'already', 'RB'),  
( 'has', 'HVZ'),  
( 'kicked', 'VBN')]
```

Exercise time!



Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Default dictionaries

- Normally, we get errors when we try to access a key in a dictionary, but we can use a special dictionary to create new entries
- `defaultdict` automatically creates an entry for this new key and give it a default value, such as 0 or the empty list
- In order to use it, we have to supply a parameter which can be used to create the default value, e.g. int, float, str, list, dict, tuple

Default dictionaries

```
# Let's import defaultdict.  
from collections import defaultdict
```

Script

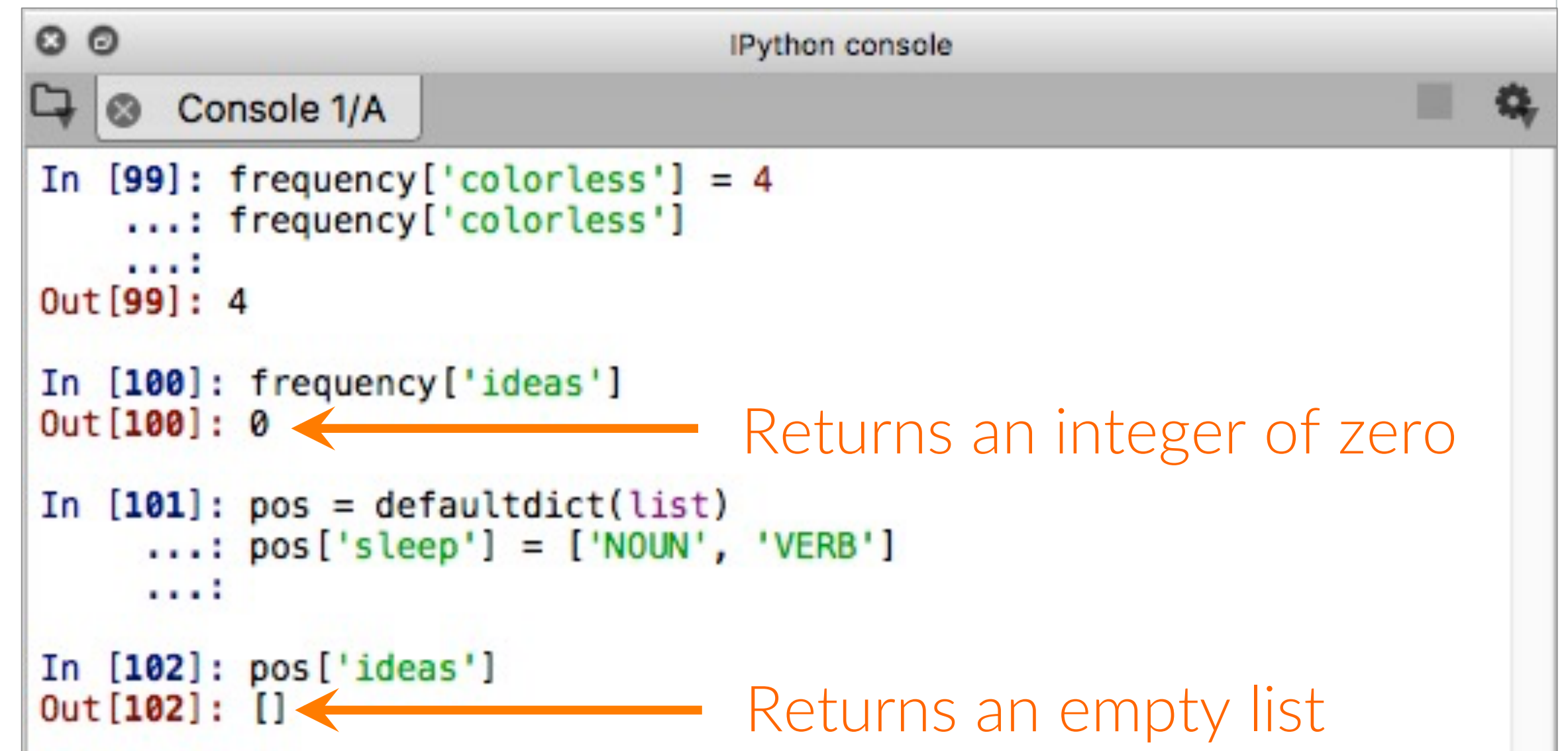
```
# Set the defaultdict to accept integers as the parameter, and name it 'frequency'.  
frequency = defaultdict(int)
```

```
# We can define the term 'colorless' to have the parameter '4'. We can check it  
# by running it - what happens when we enter a new term that's not defined?
```

```
frequency['colorless'] = 4  
frequency['colorless']  
frequency['ideas']
```

```
# Set the defaultdict to set lists as  
# the parameter, and name it 'pos'.
```

```
pos = defaultdict(list)  
pos['sleep'] = ['NOUN', 'VERB']  
pos['ideas']
```



```
IPython console  
Console 1/A  
In [99]: frequency['colorless'] = 4  
...: frequency['colorless']  
...:  
Out[99]: 4  
  
In [100]: frequency['ideas']  
Out[100]: 0 ← Returns an integer of zero  
  
In [101]: pos = defaultdict(list)  
...: pos['sleep'] = ['NOUN', 'VERB']  
...:  
  
In [102]: pos['ideas']  
Out[102]: [] ← Returns an empty list
```


Default dictionaries

- We can set a default value for new entries in our dictionary

```
# We first define the function that tells us if the POS is unknown,  
# label it as a noun.
```

```
def POS():  
    return 'NOUN'
```

```
POS()
```

```
# Let's set our default dictionary to the function  
# that will return a noun. Now, if we put in a  
# non-existent entry, it's automatically added.
```

```
pos = defaultdict(POS)  
pos['colorless'] = 'ADJ'  
pos['blog']
```

```
# We can look at the list of  
# items from the dictionary.
```

```
list(pos.items())
```

Script

```
IPython console  
Console 1/A  
In [104]: POS()  
Out[104]: 'NOUN'  
  
In [105]: pos = defaultdict(POS)  
...: pos['colorless'] = 'ADJ'  
...:  
  
In [106]: pos['blog']  
Out[106]: 'NOUN'  
  
In [107]: list(pos.items())  
Out[107]: [('colorless', 'ADJ'), ('blog', 'NOUN')]
```

Default dictionaries

- Many language processing tasks, such as tagging, have trouble identifying words that only appear once in a text (also known as **hapaxes**)
- They can perform better with a fixed vocabulary and a guarantee that no new words will appear.
- We can create a token, UNK, as a special 'out of vocabulary' token and assign it to all the low-frequency words

Default dictionaries

- We can set a default value for new entries in our dictionary

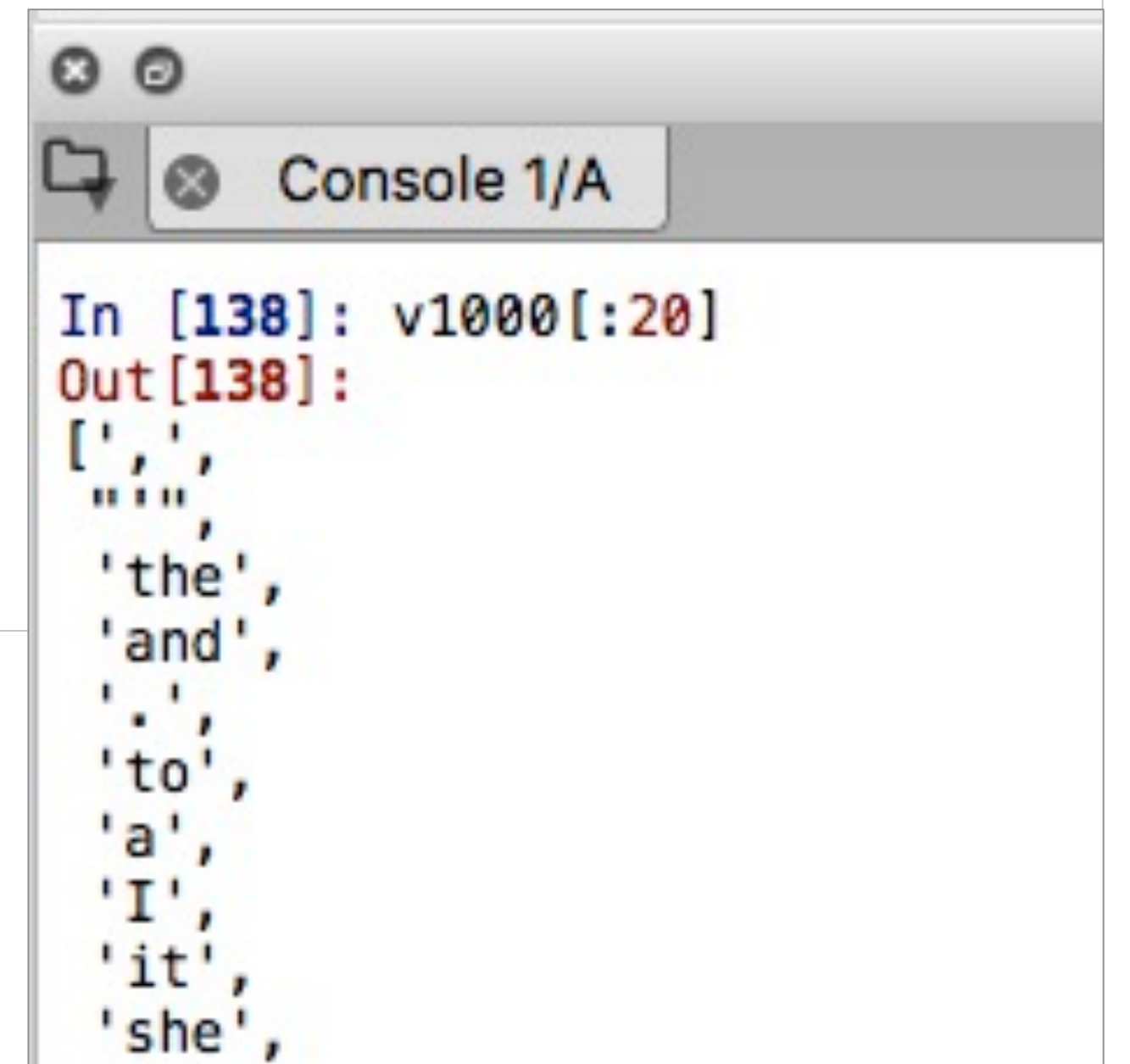
```
# Let's read in "Alice in Wonderland" and save it as 'alice'.  
alice = nltk.corpus.gutenberg.words('carroll-alice.txt')
```

```
# We'll calculate the Frequency Distribution of words, and then subset out the  
# 1000 most common words.
```

```
vocab = nltk.FreqDist(alice)  
v1000 = [word for (word, _) in vocab.most_common(1000)]
```

```
# Let's check the first 20 words of the set we just created.  
v1000[:20]
```

Script



Console 1/A

```
In [138]: v1000[:20]  
Out[138]:  
['',  
'',  
'the',  
'and',  
'',  
'to',  
'a',  
'I',  
'it',  
'she',
```


Default dictionaries

- We can set a default value for new entries in our dictionary

```
# We need to define a new function to automatically return the special  
# token that we can define - let's call it 'UNK'.  
def default_POS():  
    return 'UNK'  
  
# Set the defaultdict to return our special token and label it 'mapping'. Then,  
# we will map 'v' to itself to assign  
mapping = defaultdict(default_POS)  
for v in v1000:  
    mapping[v] = v  
mapping  
  
# Now, we'll use the mapping syntax again, but this time for all of Alice  
# - now it will assign 'UNK' to any word that is not in the top 1000 common words.  
alice2 = [mapping[v] for v in alice]  
alice2[:100]
```

Script

Exercise time!



Default dictionaries

- We can add new occurrences incrementally to dictionaries

```
# First, set the default dictionary for integers, and import brown.
```

Script

```
counts = defaultdict(int)
```

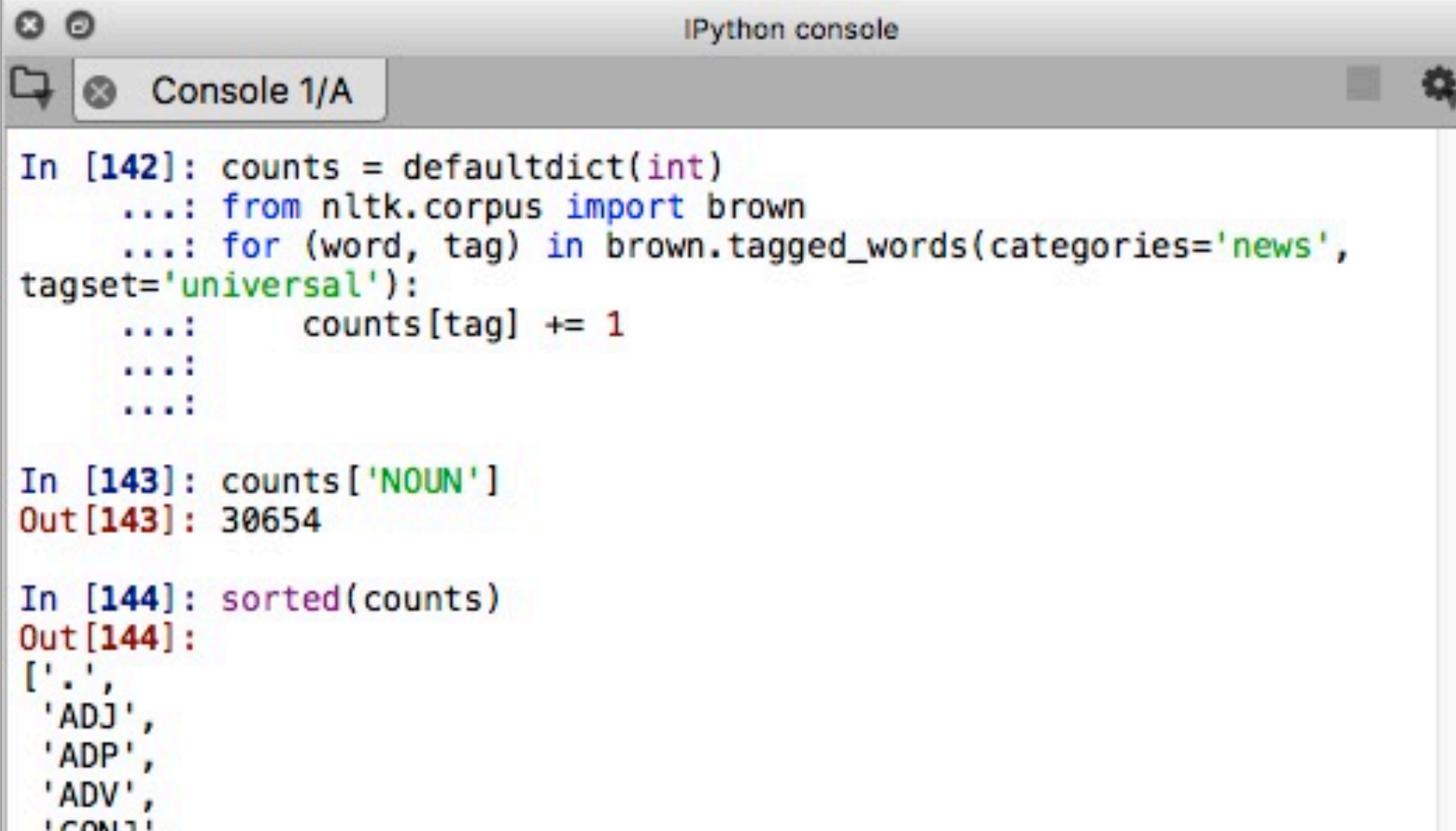
```
from nltk.corpus import brown
```

```
# If the tag hasn't been seen before, it will have a zero count by default. Each time  
# we encounter a tag, we increment its count using the += operator.
```

```
for (word, tag) in brown.tagged_words(categories = 'news', tagset = 'universal'):
```

```
    counts[tag] += 1
```

```
counts['NOUN']
```



```
IPython console
Console 1/A

In [142]: counts = defaultdict(int)
...: from nltk.corpus import brown
...: for (word, tag) in brown.tagged_words(categories='news',
...: tagset='universal'):
...:     counts[tag] += 1
...:
...:
...:

In [143]: counts['NOUN']
Out[143]: 30654

In [144]: sorted(counts)
Out[144]:
['.',
 'ADJ',
 'ADP',
 'ADV',
 'CONJ',
```


Default dictionaries

- Itemgetter is important in helping us sort a dictionary by its values

```
# First, import itemgetter.
```

Script

```
from operator import itemgetter
```

Items to sort – a list of tuples
with POS tag and frequency

Returns a callable object
that fetches an item

Tells us that the item
should be returned in
decreasing order

```
sorted(counts.items(), key = itemgetter(1), reverse = True)
```

```
# Let's set pair equal to a tuple and look at how we can pull out a particular  
# object from the tuple.
```

```
pair = ('NP', 8336)
```

```
pair[1]
```

```
itemgetter(1)(pair)
```

Exercise time!



Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Default dictionaries

- Let's review the concept of default dictionaries:

```
my_dictionary = defaultdict(function to create default value)
```

Sets the default value

```
for item in sequence:  
    my_dictionary[item_key] is updated with information about item
```

Cycle through the data set and update information

Default dictionaries

- Let's index words according to their last two letters.

```
# First, set 'last_letters' as a default dictionary for lists  
last_letters = defaultdict(list)
```

```
# Set the wordlist to the NLTK corpus in English  
words = nltk.corpus.words.words('en')
```

```
# Build the for loop where the key is equal to the last  
# two letters, and then, we'll append the words  
# to the matching key (which is the last two letters)
```

```
for word in words:  
    key = word[-2:]  
    last_letters[key].append(word)
```

```
# Let's try out our function, and pull the top ten terms!  
last_letters['ly'][:10]  
last_letters['zy'][:10]
```

Script



```
IPython  
Console 1/A  
In [159]: last_letters['ly'][:10]  
Out[159]:  
['abactinally',  
'abandonedly',  
'abasedly',  
'abashedly',  
'abashlessly',  
'abbreviately',  
'abdominally',  
'abhorrently',  
'abidingly',  
'abiogenetically']  
  
In [160]: last_letters['zy'][:10]  
Out[160]:  
['blazy',  
'bleezy',  
'blowzy',  
'boozy',  
'breezy',  
'bronzzy',  
'buzzy',  
'Chazy',  
'cozy',  
'crazy']
```

Default dictionaries: anagrams

- Let's index words according to their last two letters.

```
# First, set 'last_letters' as a default dictionary for lists.  
anagrams = defaultdict(list)
```

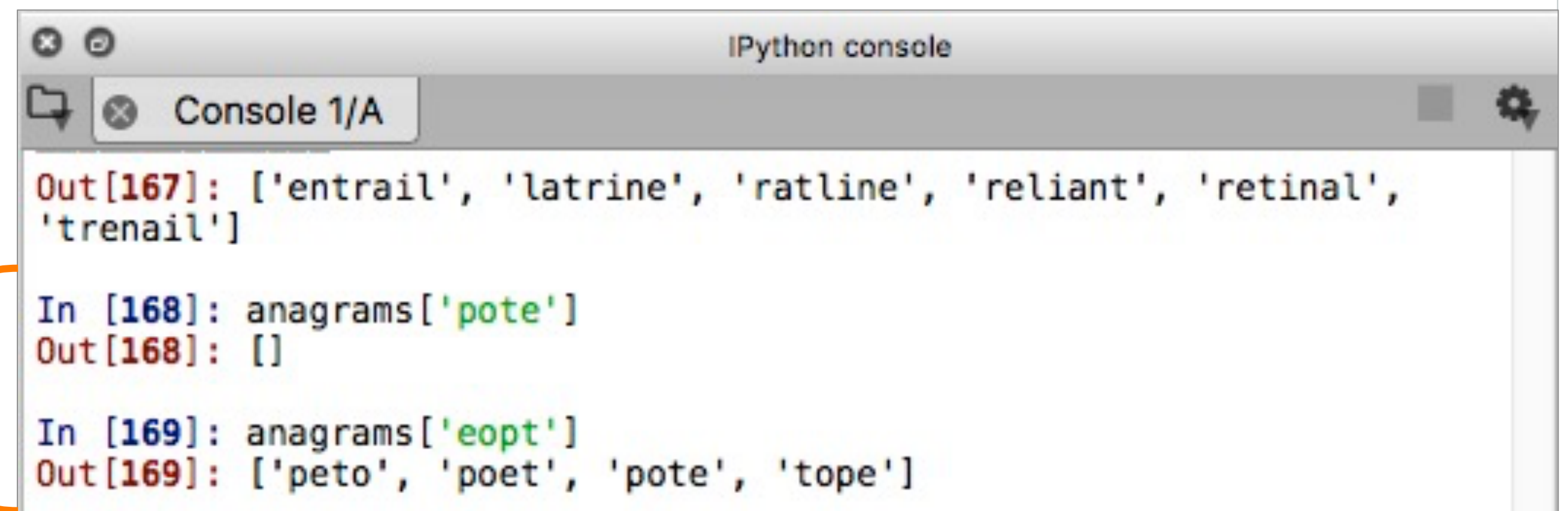
Script

```
# Let's set a for loop, where the key is equal the sorted letters in a word, and  
# the event is equal to the word itself - this way, for every word that has the  
# same letters, it will have the same key.
```

```
for word in words:  
    key = ''.join(sorted(word))  
    anagrams[key].append(word)
```

```
# Let's test it out.  
anagrams['aeilnrt']
```

```
# Note: this will only work if we put  
# the letters in alphabetical order!
```



The screenshot shows an IPython console window with the following output:

```
Out[167]: ['entrail', 'latrine', 'ratline', 'reliant', 'retinal',  
'trenail']  
  
In [168]: anagrams['pote']  
Out[168]: []  
  
In [169]: anagrams['eopt']  
Out[169]: ['peto', 'poet', 'pote', 'tope']
```

An orange bracket is drawn on the left side of the console output, grouping the last three lines (In [168], Out[168], In [169], Out[169]).

Exercise time!



Default taggers

- Let's explore how to automatically add POS tags to text
- Since parts of speech can depend on the context of the sentence, we'll need to work with data at the sentence level

```
# First, set 'last_letters' as a default dictionary for lists.
from nltk.corpus import brown

# We'll set up the regular sentences
# and the tagged sentences to compare them.
brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')

# Now we will see which tag is the most common in the text.
tags = [tag for (word, tag) in brown.tagged_words(categories = 'news')]
nltk.FreqDist(tags).max()
# NN
```

Script

```
Console 1/A
In [172]: brown_tagged_sents[:1]
Out[172]:
[[('The', 'AT'),
 ('Fulton', 'NP-TL'),
 ('County', 'NN-TL'),
 ('Grand', 'JJ-TL'),
 ('Jury', 'NN-TL'),
```

```
Console 1/A
In [173]: brown_sents[:1]
Out[173]:
[['The',
 'Fulton',
 'County',
 'Grand',
 'Jury',
```


Default taggers

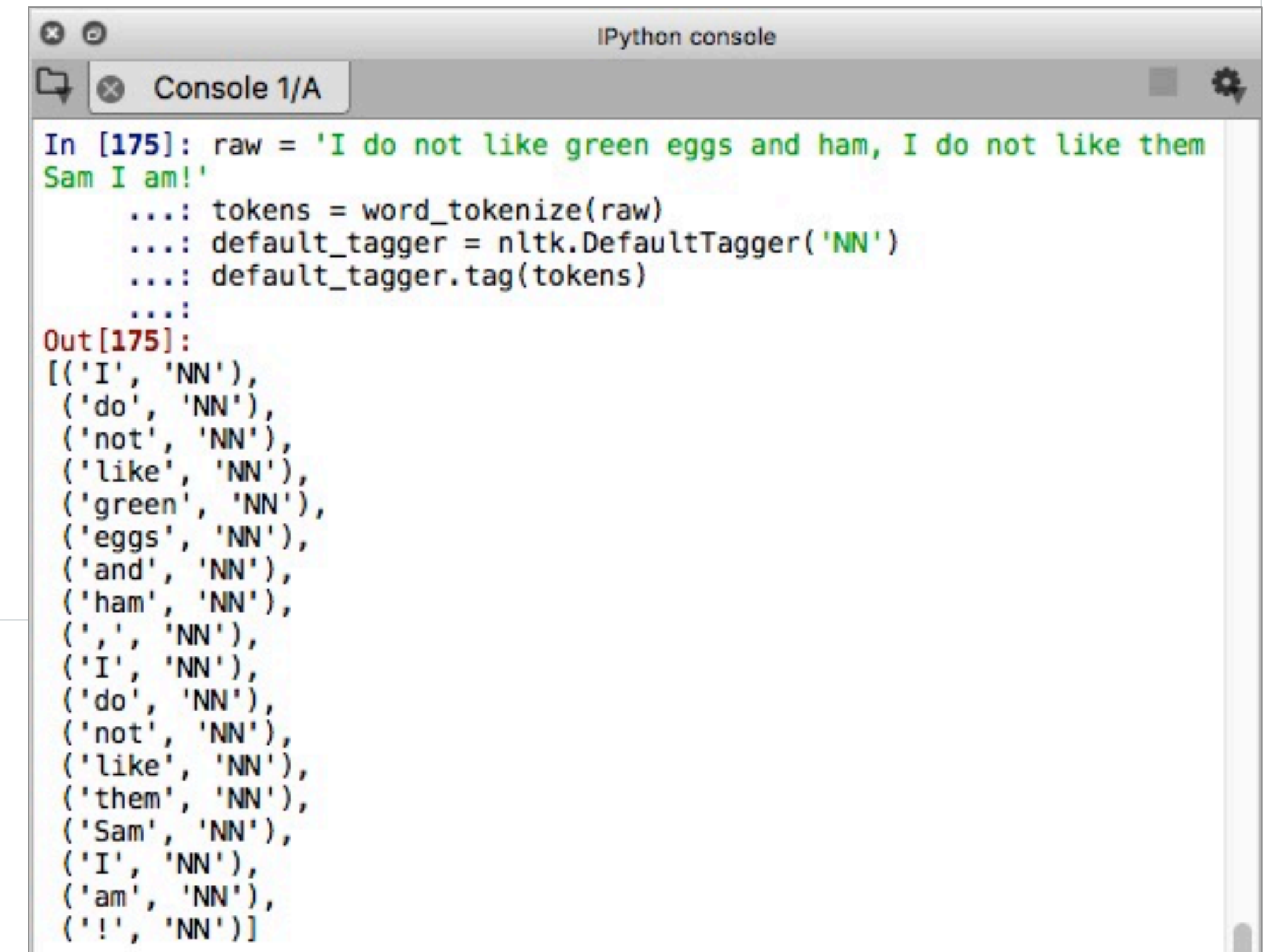
```
# Now we can create a tagger that will tag all words as 'NN'.  
raw = 'I do not like green eggs and ham, I do not like them Sam I am!'
```

Script

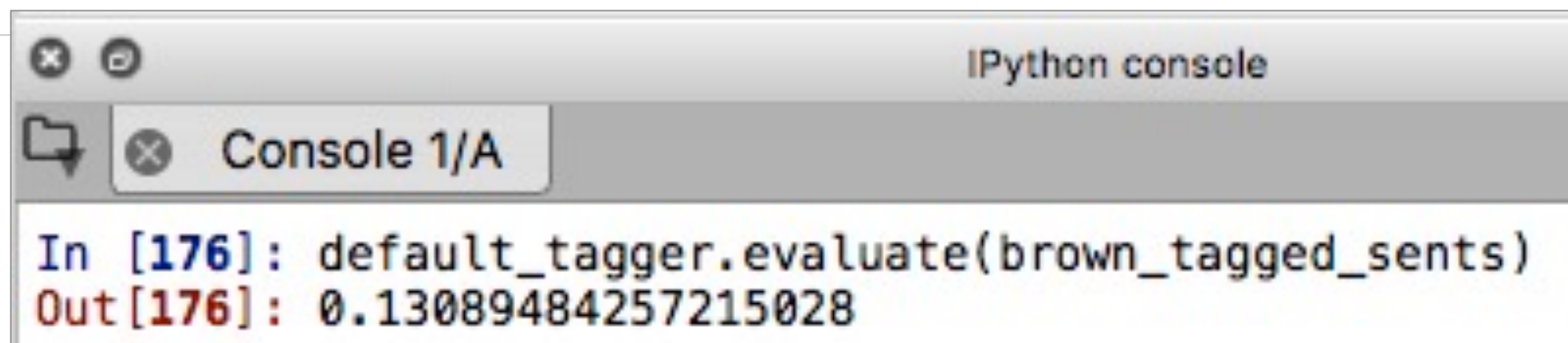
```
# We'll tokenize the words in the sentence and then use the default tagger to 'NN'.  
tokens = word_tokenize(raw)  
default_tagger = nltk.DefaultTagger('NN')
```

```
# Let's check to make sure it worked.  
default_tagger.tag(tokens)
```

```
# Now we can evaluate the method  
default_tagger.evaluate(brown_tagged_sents)
```



```
IPython console  
Console 1/A  
In [175]: raw = 'I do not like green eggs and ham, I do not like them  
Sam I am!'  
...: tokens = word_tokenize(raw)  
...: default_tagger = nltk.DefaultTagger('NN')  
...: default_tagger.tag(tokens)  
...:  
Out[175]:  
[('I', 'NN'),  
( 'do', 'NN'),  
( 'not', 'NN'),  
( 'like', 'NN'),  
( 'green', 'NN'),  
( 'eggs', 'NN'),  
( 'and', 'NN'),  
( 'ham', 'NN'),  
( ',', 'NN'),  
( 'I', 'NN'),  
( 'do', 'NN'),  
( 'not', 'NN'),  
( 'like', 'NN'),  
( 'them', 'NN'),  
( 'Sam', 'NN'),  
( 'I', 'NN'),  
( 'am', 'NN'),  
( '!', 'NN')]
```



```
IPython console  
Console 1/A  
In [176]: default_tagger.evaluate(brown_tagged_sents)  
Out[176]: 0.13089484257215028
```


Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Lookup taggers

- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

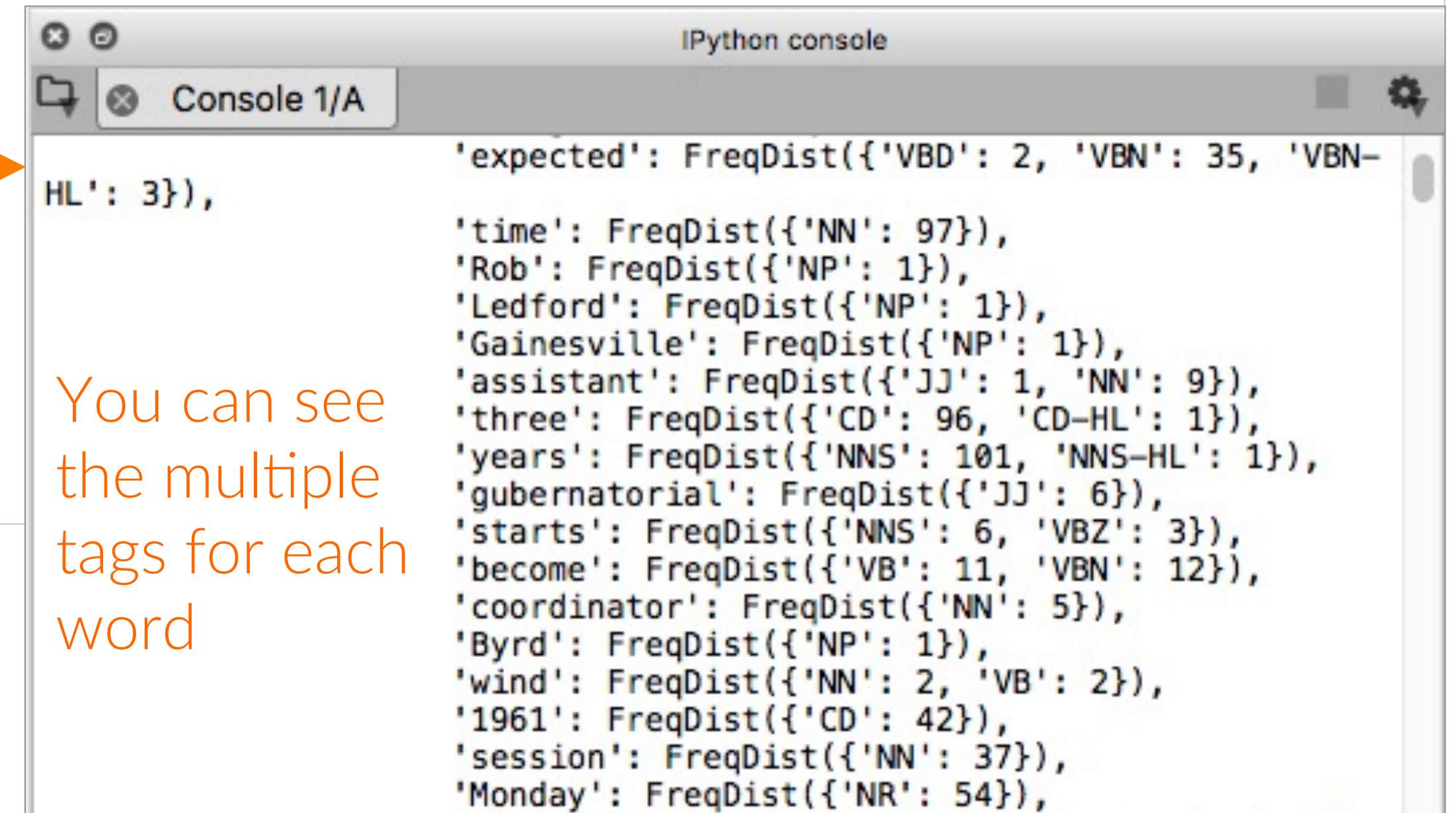
```
# First we'll get the frequency distribution and the conditional  
# frequency distribution for the Brown corpora categorized as 'news'.  
fd = nltk.FreqDist(brown.words(categories = 'news'))  
cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories = 'news'))
```

Script

```
# What does cfd look like?
```

```
cfd
```

```
# Now we will store the most common words  
# from the Brown corpus.  
corpus.most_freq_words = fd.most_common(100)
```



```
IPython console  
Console 1/A  
HL': 3}},  
'expected': FreqDist({'VBD': 2, 'VBN': 35, 'VBN-  
'time': FreqDist({'NN': 97}),  
'Rob': FreqDist({'NP': 1}),  
'Ledford': FreqDist({'NP': 1}),  
'Gainesville': FreqDist({'NP': 1}),  
'assistant': FreqDist({'JJ': 1, 'NN': 9}),  
'three': FreqDist({'CD': 96, 'CD-HL': 1}),  
'years': FreqDist({'NNS': 101, 'NNS-HL': 1}),  
'gubernatorial': FreqDist({'JJ': 6}),  
'starts': FreqDist({'NNS': 6, 'VBZ': 3}),  
'become': FreqDist({'VB': 11, 'VBN': 12}),  
'coordinator': FreqDist({'NN': 5}),  
'Byrd': FreqDist({'NP': 1}),  
'wind': FreqDist({'NN': 2, 'VB': 2}),  
'1961': FreqDist({'CD': 42}),  
'session': FreqDist({'NN': 37}),  
'Monday': FreqDist({'NR': 54}),
```

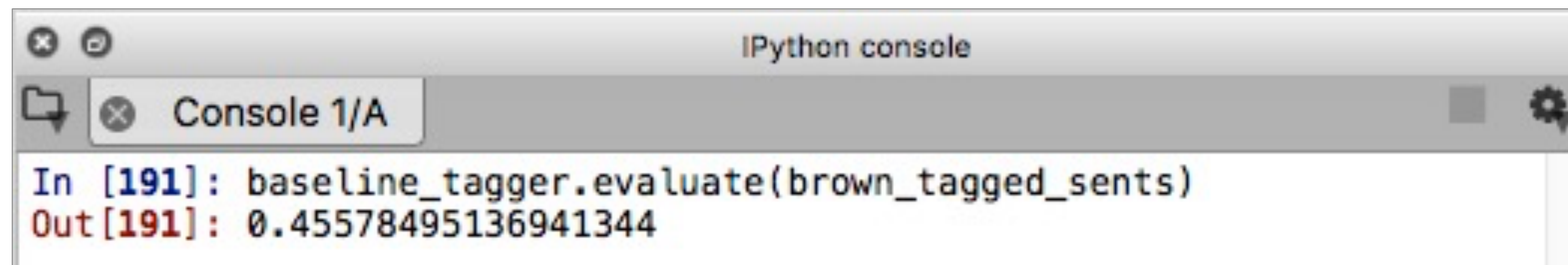
You can see the multiple tags for each word

Lookup taggers

- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

```
# We'll set 'likely_tags' equal to the max count of tags for the words that  
# are in the 'most_freq_words' data - remember, since some of the words have  
# multiple parts of speech, we only want to take the most frequent one.  
likely_tags = dict((word, cfd[word].max()) for (word, _) in most_freq_words)  
  
# Now, we can use the UnigramTagger() method and set it as the baseline tagger.  
# We can see how much more accurate we are if we tag the sentences with  
# the lookup tagger.  
baseline_tagger = nltk.UnigramTagger(model = likely_tags)  
baseline_tagger.evaluate(brown_tagged_sents)
```

Script



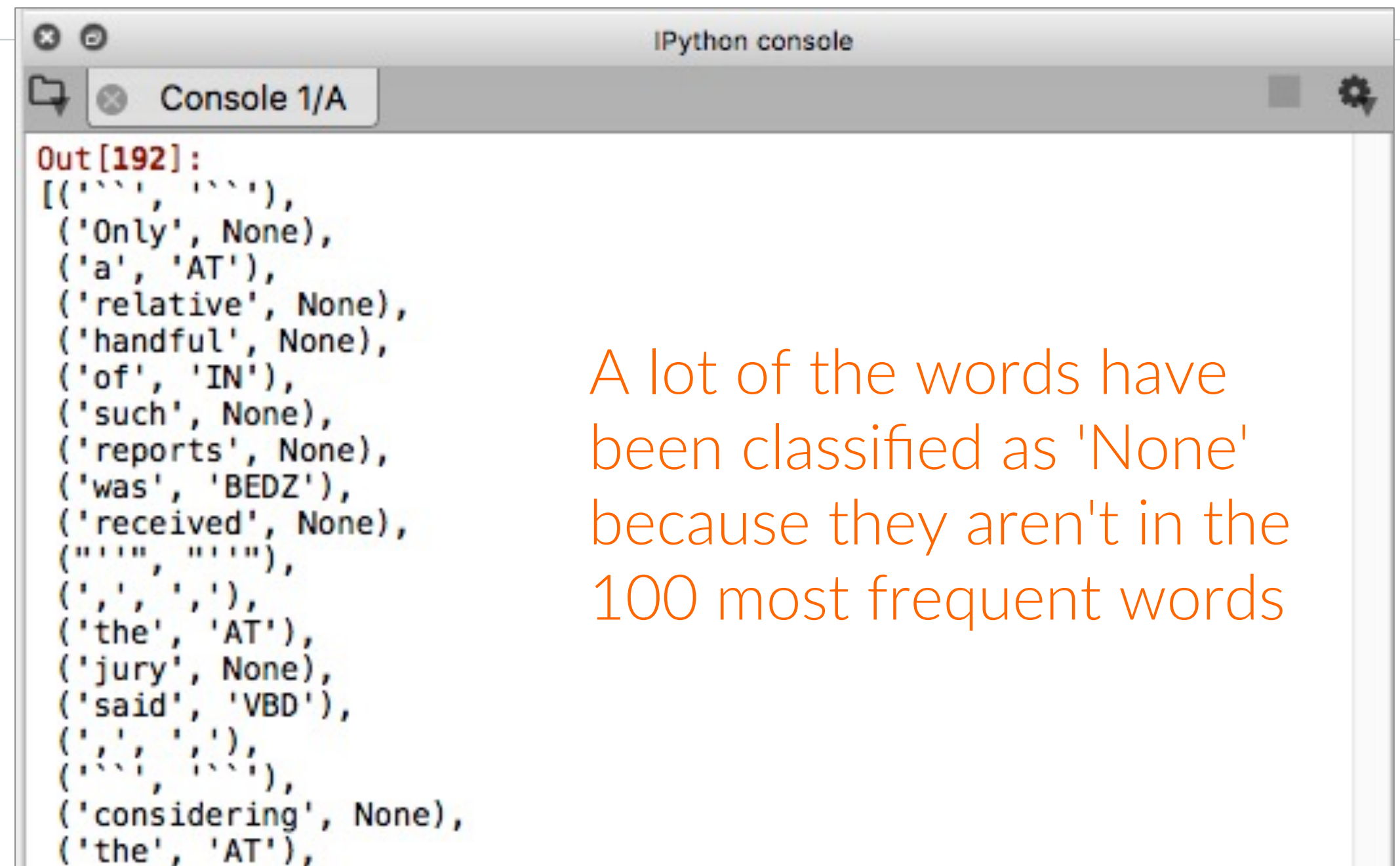
```
IPython console  
Console 1/A  
In [191]: baseline_tagger.evaluate(brown_tagged_sents)  
Out[191]: 0.45578495136941344
```


Lookup taggers

- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

```
# Let's test this out on untagged input text:  
sent = brown.sents(categories = 'news')[3]  
baseline_tagger.tag(sent)
```

Script



```
IPython console  
Console 1/A  
Out[192]:  
[('', ''),  
 ('Only', None),  
 ('a', 'AT'),  
 ('relative', None),  
 ('handful', None),  
 ('of', 'IN'),  
 ('such', None),  
 ('reports', None),  
 ('was', 'BEDZ'),  
 ('received', None),  
 ('', ''),  
 ('', ''),  
 ('the', 'AT'),  
 ('jury', None),  
 ('said', 'VBD'),  
 ('', ''),  
 ('', ''),  
 ('considering', None),  
 ('the', 'AT'),
```

A lot of the words have been classified as 'None' because they aren't in the 100 most frequent words

Lookup taggers

- We want to create a method where we want to lookup if the word is in the most frequent words, then if it isn't, assign it an 'NN' tag.

```
# This is what that method we described would look like - if the word  
# doesn't appear in the 'likely_tags' set, it will be assigned an 'NN' tag.  
baseline_tagger = nltk.UnigramTagger(model = likely_tags,  
                                     backoff = nltk.DefaultTagger('NN'))
```

Script

Lookup taggers

- Let's put this all together and write a program to evaluate lookup taggers that have a range of sizes

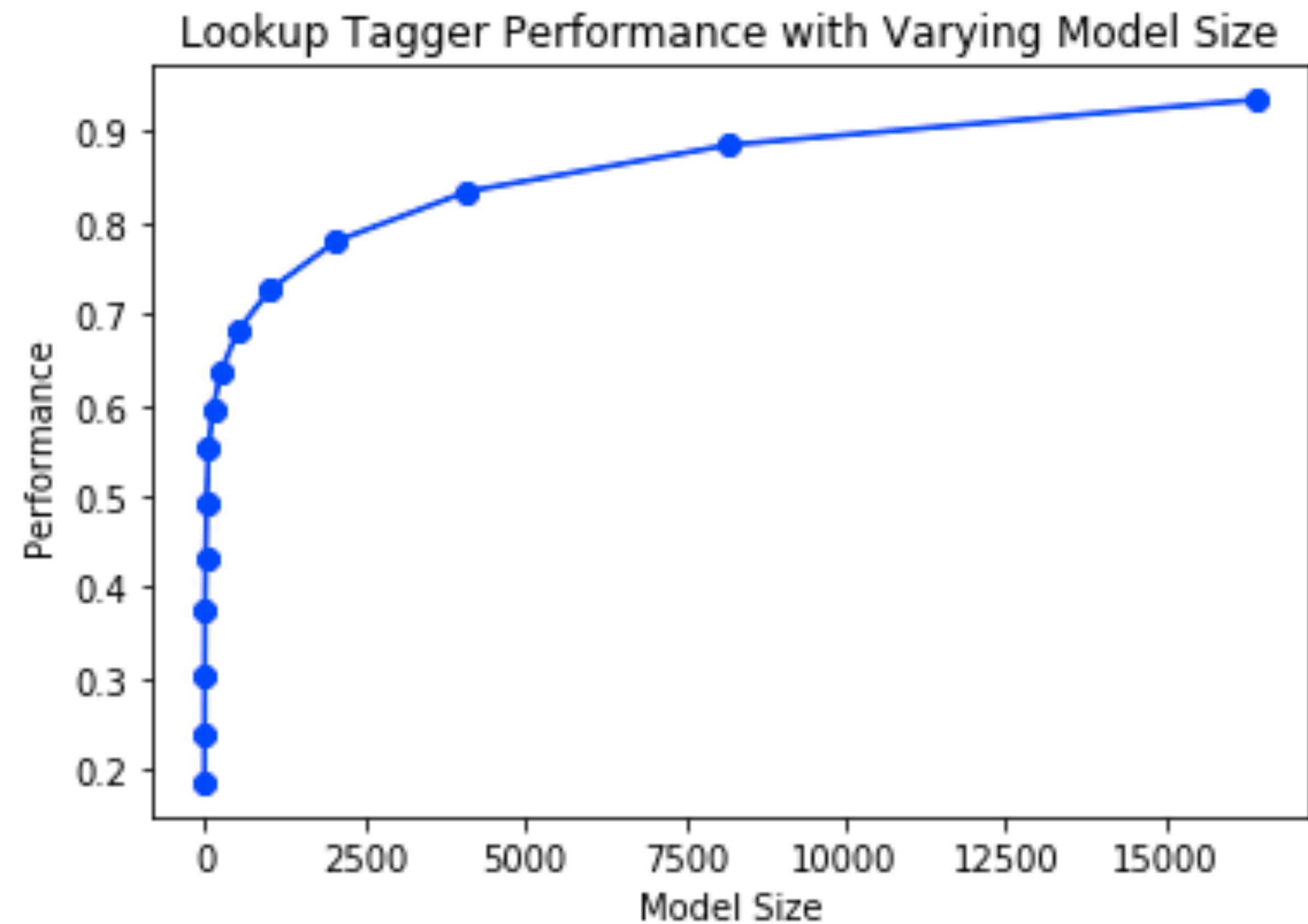
```
def performance(cfd, wordlist):  
    lt = dict((word, cfd[word].max()) for word in wordlist)  
    baseline_tagger = nltk.UnigramTagger(model = lt, backoff = nltk.DefaultTagger('NN'))  
    return baseline_tagger.evaluate(brown.tagged_sents(categories = 'news'))  
  
def display():  
    import pylab  
    word_freqs = nltk.FreqDist(brown.words(categories = 'news')).most_common()  
    words_by_freq = [w for (w, _) in word_freqs]  
    cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))  
    sizes = 2 ** pylab.arange(15)  
    perfs = [performance(cfd, words_by_freq[:size]) for size in sizes]  
    pylab.plot(sizes, perfs, '-bo')  
    pylab.title('Lookup Tagger Performance with Varying Model Size')  
    pylab.xlabel('Model Size')  
    pylab.ylabel('Performance')  
    pylab.show()
```

Script

Lookup taggers

```
display()
```

Script



Exercise time!



Unigram taggers

- A unigram tagger is based on a simple statistical algorithm based on the likelihood of the tag for a particular token.

```
# Let's import brown and set up the tagged sentences and regular sentences  
# word lists. Then, we'll train the unigram tagger, use it to tag a sentence,  
# and evaluate it.
```

```
from nltk.corpus import brown
```

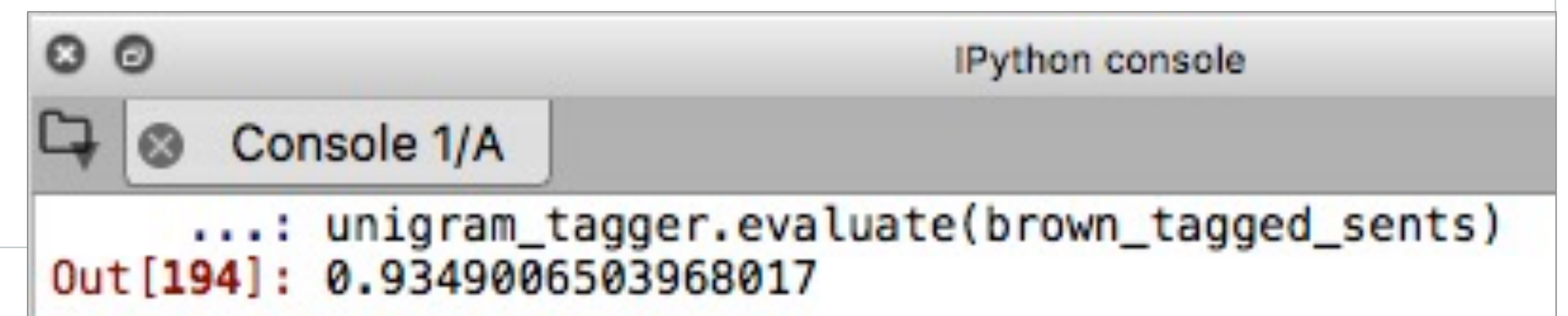
```
brown_tagged_sents = brown.tagged_sents(categories = 'news')  
brown_sents = brown.sents(categories = 'news')
```

```
# How does the unigram tagger perform on the sentences of 2007?
```

```
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)  
unigram_tagger.tag(brown_sents[2007])
```

```
unigram_tagger.evaluate(brown_tagged_sents)
```

Script



The screenshot shows a terminal window titled "IPython console" with a sub-tab "Console 1/A". It displays the execution of the final line of code from the script: `...: unigram_tagger.evaluate(brown_tagged_sents)`. The output is shown in red text: `Out[194]: 0.9349006503968017`.

Separating training and testing data

- In order to make sure that our model is generalizable to new data, we have to train and test it. Let's split the data to train 90% of it, and test 10% of it.

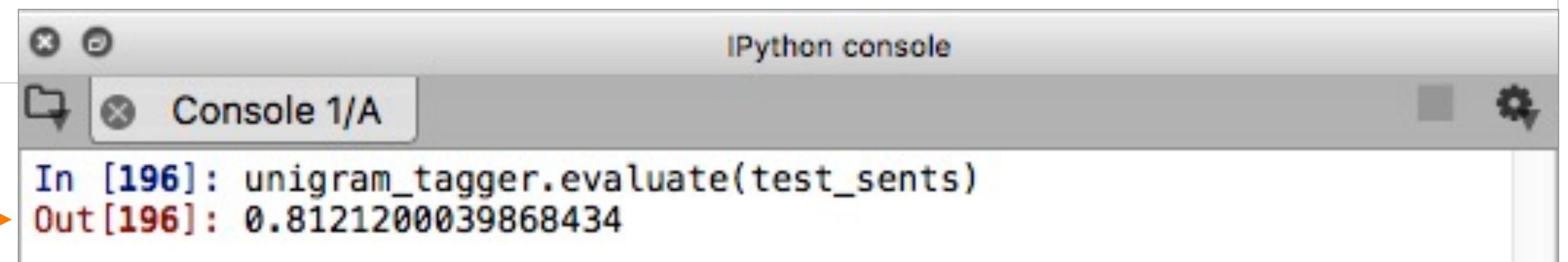
```
# We can split up the data to 90% of the tagged sentences to train.
size = int(len(brown_tagged_sents) * 0.9)
size

# Now, we will label a data set 'train_sents' and 'test_sents' to train the model.
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]

# We'll train our model with UnigramTagger, and then test it with 'evaluate'.
unigram_tagger = nltk.UnigramTagger(train_sents)
unigram_tagger.evaluate(test_sents)
```

Script

Even though the performance is worse,
we have a better picture about the
usefulness of this tagger →



The screenshot shows an IPython console window titled "IPython console" with a sub-tab "Console 1/A". It displays the execution of the command `unigram_tagger.evaluate(test_sents)` in the prompt `In [196]:`, resulting in the output `Out[196]: 0.8121200039868434`.