Coding & World Language Lit: Analyzing Dante’s Inferno with Python NLTK

This beginner-level tutorial shows you how to analyze a foreign language text using Python NLTK (Natural Language Toolkit), a Natural Language Processing platform which allows you to break down words into their stems and determine the most commonly used words in a text.

Computer science can connect to any other discipline — so if you’re curious about programming *and* dream of life in an Italian villa, this tutorial is for you!

In this tutorial, we’re going to analyze the text of Dante’s *Inferno* through its word frequencies. *Inferno* is a 14th-century Italian poem that details the author’s fictitious trip to hell. Dante, the author, is guided by the Roman poet Vergil, who shows him around the nine circles of hell.

Although it has been translated into English, the original poem is written in Italian. Can you anticipate any problems with analyzing word frequency in an Italian poem?

Consider verb conjugations in Romance languages vs. English. In English, you can say “I speak” and “you speak,” and the word “speak” doesn’t change. The subject (the person who’s doing the action) is indicated by separate words, “I” and “you.” But in Italian — much like Spanish and other Romance languages — I would have to say *parl**o*** (I speak) and *parl**i*** (you speak), indicating the subject by the ending of the word.

So if we’re analyzing frequency of the words in Dante’s *Inferno*, will *parlo* and *parli* be counted as the same word, from the infinitive *parlare*? No, because computers are stupid and don’t know Italian. Luckily, there is a solution!

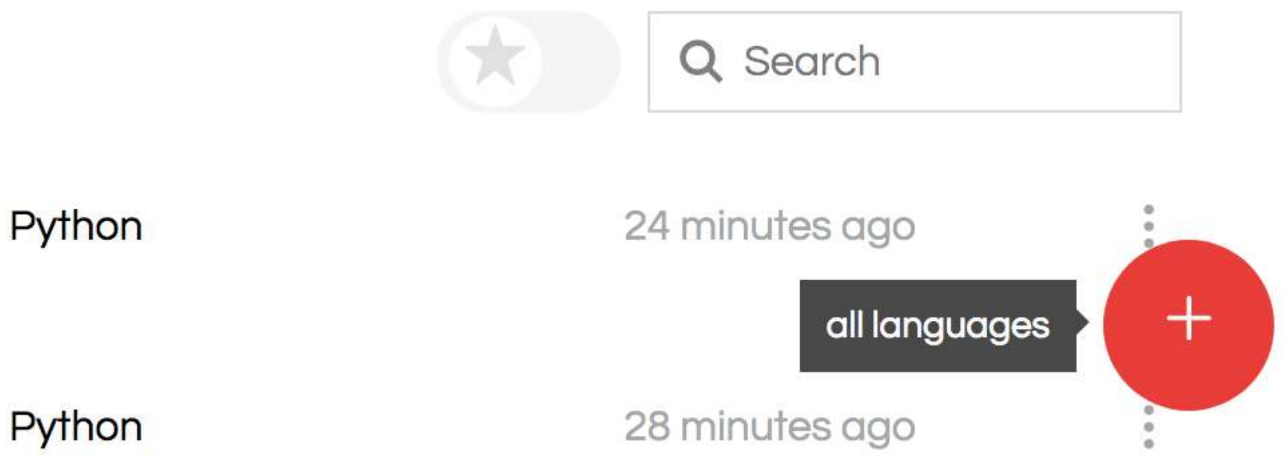
Before we analyze the text, we can separate all of the words using a process called **tokenization**. Then we break down each word into its stem. This process would transform *parlo* and *parli* into their present stem, *parl-*, allowing them to be counted as the same word. At that point, the text would be ready for analysis.

Let’s get started programming in Python so we can tokenize Dante’s *Inferno*!

Getting started with Python

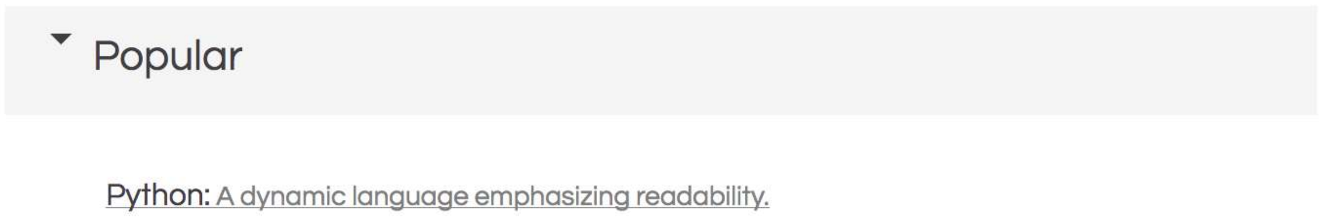
Python is overall an awesome language for data analysis, whether we're more science-focused or humanities-focused.

To start programming with Python right in the web browser, go to repl.it and make an account. After that, click on the red plus sign in the bottom right corner of the screen to make a coding environment:

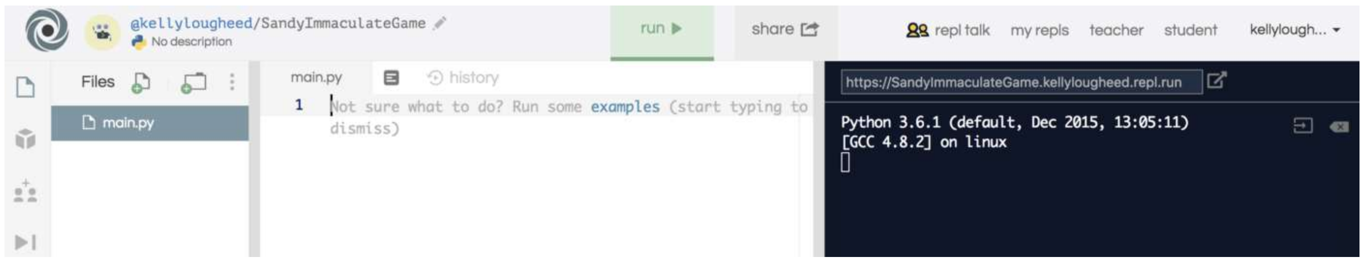


Click on the red plus sign.

Select "Python" from the "Popular" category:



Then you should arrive at a blank Python editor:



```
Not sure what to do? Run some examples (start typing to
dismiss)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

You’ll be typing in the white text area.

First, we need to import and download everything we need from NLTK and other packages. Copy and paste these imports into your Python editor, and then read the following explanation.

```
import nltk
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import word_tokenize
from urllib import request


nltk.download('punkt')
nltk.download('stopwords')
```

Our imports:

- The Natural Language Toolkit that comes with Python
- Stop words from NLTK. These are common words like “a” and “the” which we’ll want to filter out when we analyze frequency. Stop words are available for many languages, including Italian.
- The Snowball Stemmer from NLTK. This tool will let us reduce every word to its stem so we can accurately count its frequency regardless of its grammatical context. The Snowball Stemmer works with foreign languages.
- A word tokenizing function, which allows us to break up texts into lists of words for processing.
- A request function, which allows Python to read text from URLs.

Finally, we download a package called “punkt” from NLTK that allows us to use the word tokenizing function, and another package called “stopwords” that allows us to access the stop words mentioned above.

Final stylistic note: It's good practice to alphabetize your imports.

Importing text from a URL

It's really hard to copy and paste the entire text of Dante's *Inferno* into a separate file (I tried), so instead we're going to read the text off a website using Python!

I located the .txt file (generally easiest to deal with) of *Inferno* at `https://www.gutenberg.org/cache/epub/997/pg997.txt`, so the first thing I'm going to do (below my imports) is store that URL in a variable:

```
url = "https://www.gutenberg.org/cache/epub/997/pg997.txt"
```

Now, whenever I say `url`, the computer will know I mean that gutenberg.org address.

Next, I can use the request method I imported to send a request to gutenberg.org's server and get the text of *Inferno* when the server responds:

```
response = request.urlopen(url)
```

The server will deliver the *Inferno* text and store it in the `response` variable, but its format will be a little more complex than we need. We can turn the *Inferno* into an easy-to-read string of text with the following line of code:

```
text = response.read().decode('utf8')
```

Basically, I've just told the computer to translate the overly-complex variable `response`into a simple string of text using UTF-8, a common form of character encoding. Now the variable `text` will hold a bunch of text in a simple format like this:

```
"Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che
la diritta via era smarrita..."
```

Basically, the entire text of *Inferno* surrounded by quotation marks. In computer science, we call this data type (a bunch of characters/words/phrases surrounded by quotation marks) a string.

Okay, so I've got the text in a friendly format. What's the next thing to do? If you said "tokenize the text," you're correct!

Tokenizing the text

Previously, I mentioned that tokenization resulted in "lists of words for processing." A list is actually a data structure in Python (similar to an array in other languages). Whereas a variable looks like this:

```
favorite_food = "tiramisu"
```

A list looks like this:

```
grocery_list = ["pasta", "tomato sauce", "pesto", "gelato"]
```

You can see that while a variable contains just one thing, a list contains many things, making it useful for grocery lists and also lists of words from books. When we tokenize *Inferno*, it'll go from looking like this:

```
"Nel mezzo del cammin di nostra vita..."
```

To looking like this:

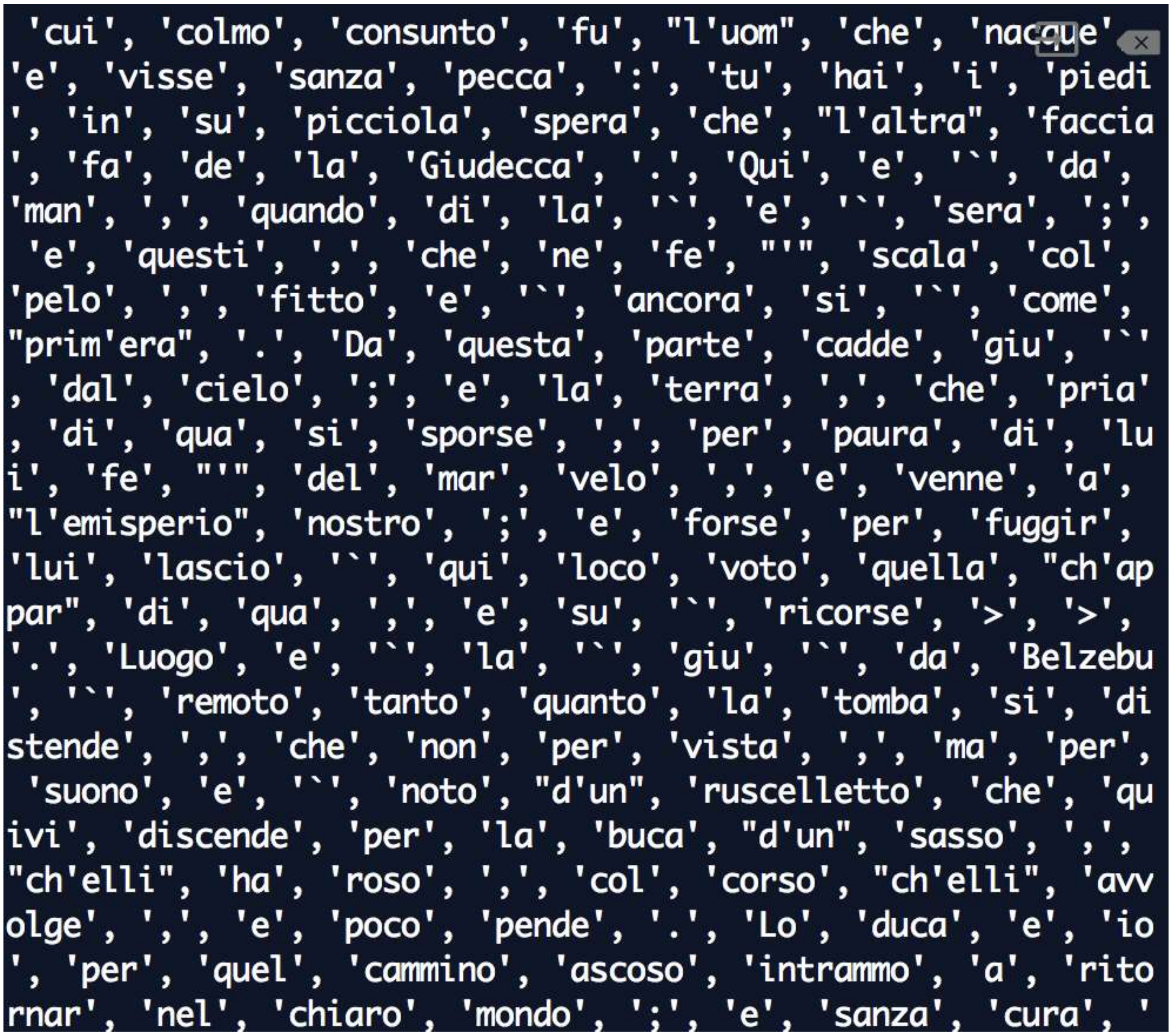
```
["Nel", "mezzo", "del", "cammin", "di", "nostra", "vita"]
```

Writing the source code for tokenization can be complex, but luckily, NLTK supplies that function for us! To convert the raw text of *Inferno* into tokens, write this line of code:

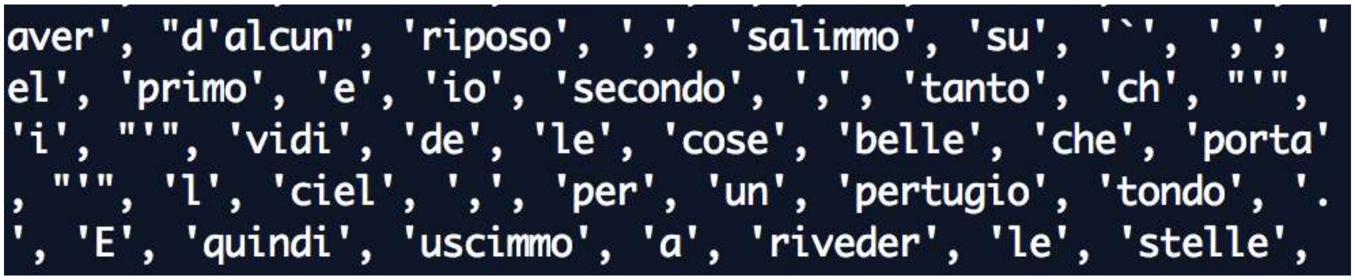
```
tokens = word_tokenize(text)
```

Now the variable `tokens` should contain a list of words. If you want to see what this list looks like (or at the least the bottom of it, as repl.it can only print so much), you can print the tokens like so:

```
print(tokens)
```



```
 'cui', 'colmo', 'consunto', 'fu', "l'uom", 'che', 'nacque'
'e', 'visse', 'sanza', 'pecca', ':', 'tu', 'hai', 'i', 'piedi
', 'in', 'su', 'picciola', 'spera', 'che', "l'altra", 'faccia
', 'fa', 'de', 'la', 'Giudecca', '.', 'Qui', 'e', '`', 'da',
'man', ',', 'quando', 'di', 'la', '`', 'e', '`', 'sera', ';',
 'e', 'questi', ',', 'che', 'ne', 'fe', "'", 'scala', 'col',
'pelo', ',', 'fitto', 'e', '`', 'ancora', 'si', '`', 'come',
"prim'era", '.', 'Da', 'questa', 'parte', 'cadde', 'giu', '`'
, 'dal', 'cielo', ';', 'e', 'la', 'terra', ',', 'che', 'pria'
, 'di', 'qua', 'si', 'sporse', ',', 'per', 'paura', 'di', 'lu
i', 'fe', "'", 'del', 'mar', 'velo', ',', 'e', 'venne', 'a',
"l'emisperio", 'nostro', ';', 'e', 'forse', 'per', 'fuggir',
'lui', 'lascio', '`', 'qui', 'loco', 'voto', 'quella', "ch'ap
par", 'di', 'qua', ',', 'e', 'su', '`', 'ricorse', '>', '>',
'.', 'Luogo', 'e', '`', 'la', '`', 'giu', '`', 'da', 'Belzebu
', '`', 'remoto', 'tanto', 'quanto', 'la', 'tomba', 'si', 'di
stende', ',', 'che', 'non', 'per', 'vista', ',', 'ma', 'per',
 'suono', 'e', '`', 'noto', "d'un", 'ruscelletto', 'che', 'qu
ivi', 'discende', 'per', 'la', 'buca', "d'un", 'sasso', ',',
"ch'elli", 'ha', 'roso', ',', 'col', 'corso', "ch'elli", 'avv
olge', ',', 'e', 'poco', 'pende', '.', 'Lo', 'duca', 'e', 'io
', 'per', 'quel', 'cammino', 'ascoso', 'intrammo', 'a', 'rito
rnar', 'nel', 'chiaro', 'mondo', ';', 'e', 'sanza', 'cura', '
```



```
aver', "d'alcun", 'riposo', ',', 'salimmo', 'su', '`', ',', '
el', 'primo', 'e', 'io', 'secondo', ',', 'tanto', 'ch', "'",
'i', "'", 'vidi', 'de', 'le', 'cose', 'belle', 'che', 'porta'
, "'", 'l', 'ciel', ',', 'per', 'un', 'pertugio', 'tondo', '.
', 'E', 'quindi', 'uscimmo', 'a', 'riveder', 'le', 'stelle',
```

The tokens!

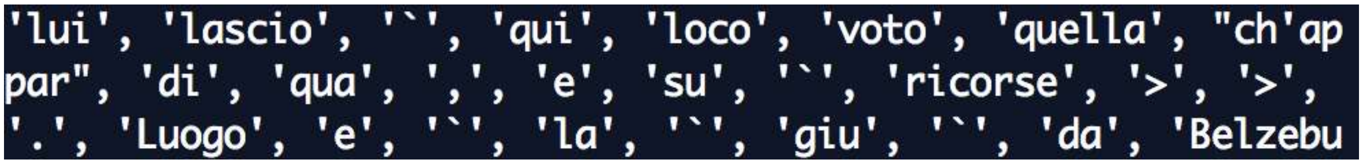
The computer might take a little bit of time to produce the tokens. Bask in your achievements so far as you wait for the computer — computers ordinarily work so fast that whenever you make them slow, you know you're doing complicated, important things (or accidentally wrote an infinite loop).

As a shortcut, I like to just print the length of the tokens list, which is to say, the number of words in my list. You can do that using the `len` function, which calculates the length of a list:

```
print(len(tokens))
```

You should have 45,274 tokens (or words)!

If you printed the tokens, you'll notice that punctuation is also included. What?!



```
'lui', 'lascio', '`', 'qui', 'loco', 'voto', 'quella', "ch'ap
par", 'di', 'qua', ',', 'e', 'su', '`', 'ricorse', '>', '>',
'.', 'Luogo', 'e', '`', 'la', '`', 'giu', '`', 'da', 'Belzebu
```

Some of the tokens are just commas and accents!

Luckily, we can get rid of this madness with the line of code below:

```
tokens = [token for token in tokens if token not in '.,:;<>!?[]()`"\'']
```

Woah! What is going on?! Basically, this is a cool programming maneuver where I said: "Set the `tokens` variable equal to the same thing as previously (all the tokens that were there before), but filter out any token that is actually just any of the punctuation marks that I've explicitly listed out!"

If you print the length of tokens again, you should now have 34,836 tokens.

You might notice in that among the tokens, we have basic words like "la" (the) and "e" (and) in there. These basic words are not interesting to us at all for analyzing word frequency — who cares if Dante said "the" 1000 times?!

These basic words are called **stop words** in NLTK, and we can filter them out.

Filtering out the stop words

Now we're ready to get rid of the stop words, the super common words that we don't care about. First, let's store the stop words in a variable:

```
stop_words = stopwords.words("italian")
```

Now, we can do some fancy programming footwork to filter out the steps from our list. Here's how we reset the `tokens` variable to a list of words without the stop words:

```
tokens = [token for token in tokens if token not in stop_words]
```

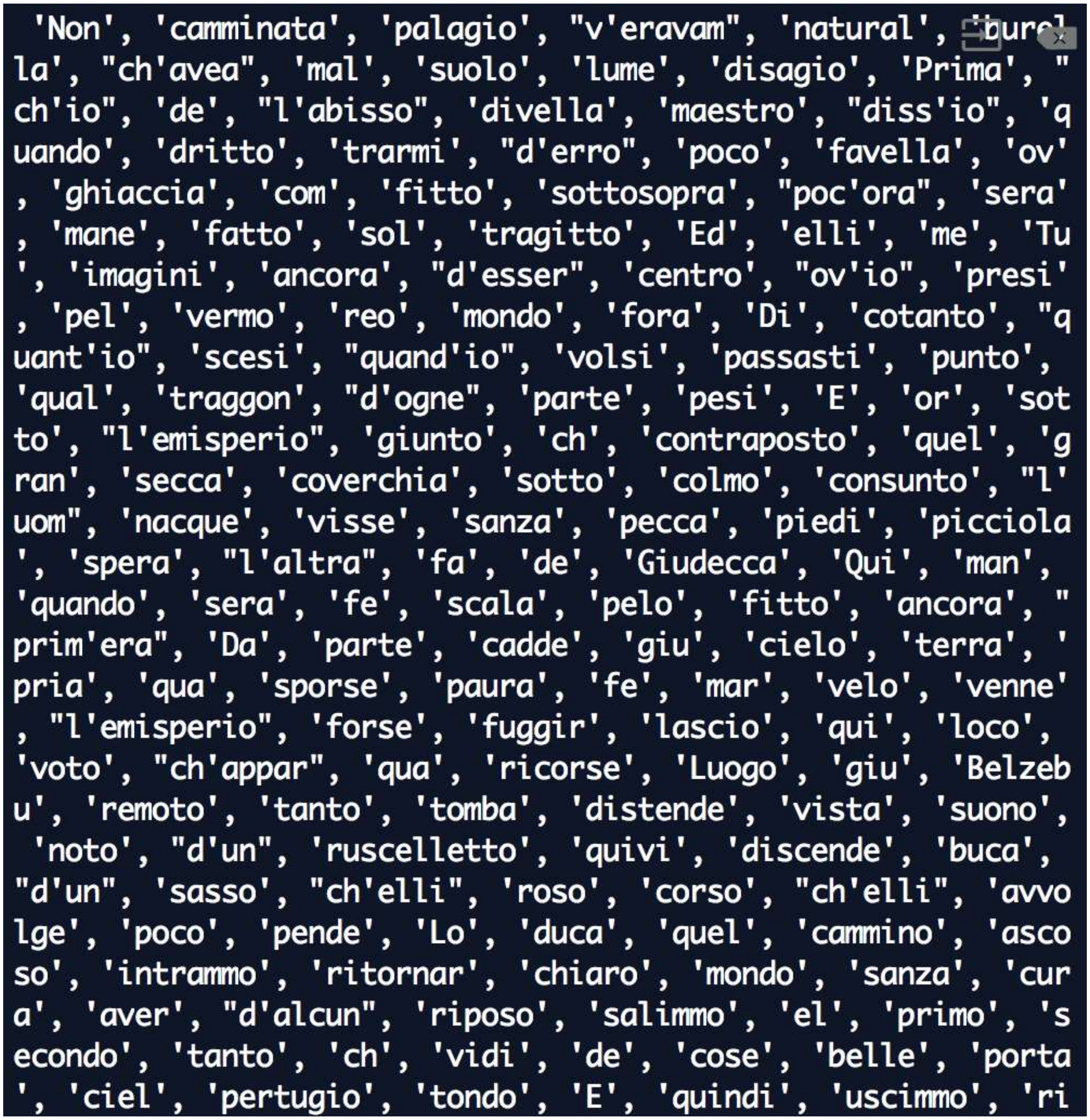
Basically, I said, "Make the `tokens` variable equal to all the same tokens as before, except for the words/tokens in the `stop_words` list!"

Just to double-check that all those pesky stop words are gone, we can print out the tokens — and also the length of tokens, to make sure it's different than before!

```
print(tokens)
print(len(tokens))
```

The `tokens` list should now hold 21,781 words, wayyy down from our previous 34k or so words.

We can also see that our new list of tokens is happily devoid of filler words and punctuation:



```
 'Non', 'camminata', 'palagio', "v'eravam", 'natural', 'burel
la', "ch'avea", 'mal', 'suolo', 'lume', 'disagio', 'Prima', "
ch'io", 'de', "l'abisso", 'divella', 'maestro', "diss'io", 'q
uando', 'dritto', 'trarmi', "d'erro", 'poco', 'favella', 'ov'
, 'ghiaccia', 'com', 'fitto', 'sottosopra', "poc'ora", 'sera'
, 'mane', 'fatto', 'sol', 'tragitto', 'Ed', 'elli', 'me', 'Tu
', 'imagini', 'ancora', "d'esser", 'centro', "ov'io", 'presi'
, 'pel', 'vermo', 'reo', 'mondo', 'fora', 'Di', 'cotanto', "q
uant'io", 'scesi', "quand'io", 'volsi', 'passasti', 'punto',
'qual', 'traggon', "d'ogne", 'parte', 'pesi', 'E', 'or', 'sot
to', "l'emisperio", 'giunto', 'ch', 'contraposto', 'quel', 'g
ran', 'secca', 'coverchia', 'sotto', 'colmo', 'consunto', "l'
uom", 'nacque', 'visse', 'sanza', 'pecca', 'piedi', 'picciola
', 'spera', "l'altra", 'fa', 'de', 'Giudecca', 'Qui', 'man',
'quando', 'sera', 'fe', 'scala', 'pelo', 'fitto', 'ancora', "
prim'era", 'Da', 'parte', 'cadde', 'giu', 'cielo', 'terra', '
pria', 'qua', 'sporse', 'paura', 'fe', 'mar', 'velo', 'venne'
, "l'emisperio", 'forse', 'fuggir', 'lascio', 'qui', 'loco',
'voto', "ch'appar", 'qua', 'ricorse', 'Luogo', 'giu', 'Belzeb
u', 'remoto', 'tanto', 'tomba', 'distende', 'vista', 'suono',
 'noto', "d'un", 'ruscelletto', 'quivi', 'discende', 'buca',
"d'un", 'sasso', "ch'elli", 'roso', 'corso', "ch'elli", 'avvo
lge', 'poco', 'pende', 'Lo', 'duca', 'quel', 'cammino', 'asco
so', 'intrammo', 'ritornar', 'chiaro', 'mondo', 'sanza', 'cur
a', 'aver', "d'alcun", 'riposo', 'salimmo', 'el', 'primo', 's
econdo', 'tanto', 'ch', 'vidi', 'de', 'cose', 'belle', 'porta
', 'ciel', 'pertugio', 'tondo', 'E', 'quindi', 'uscimmo', 'ri
```

Nice!

Stemming the words

Now we're ready to chop down our words into their stems! First order of business, we'll need to create a stemmer. You can create an Italian stemmer like so:

```
stemmer = SnowballStemmer("italian")
```

If you are working in a different language, note that you can print all the languages that the Snowball Stemmer handles with this line of code:

```
print(" ".join(SnowballStemmer.languages))
```

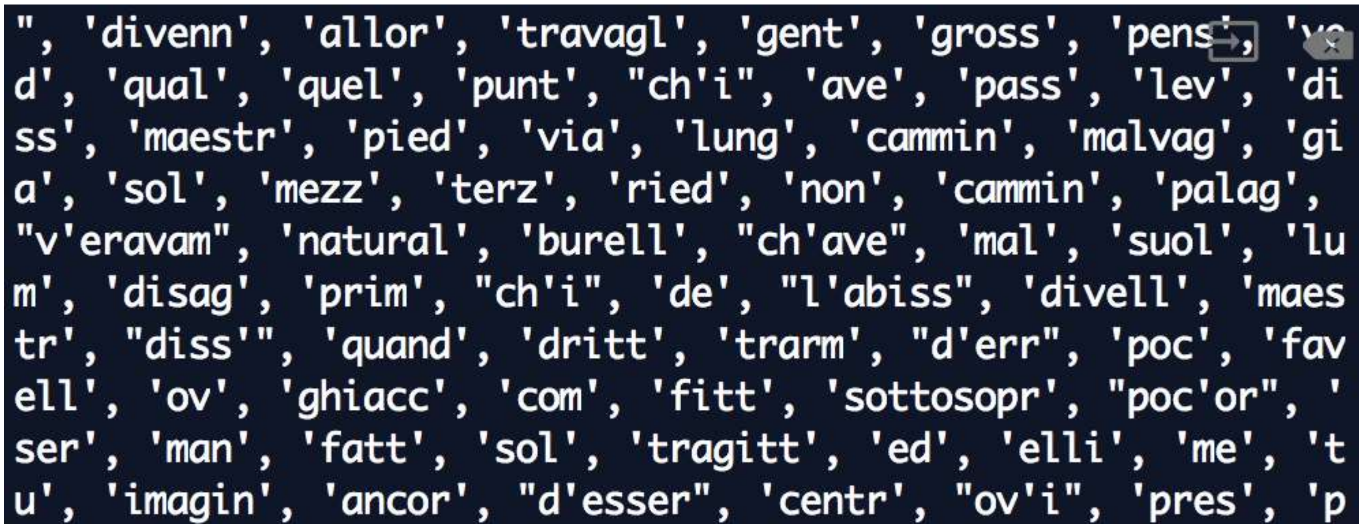
Now we can convert all the words to their stems with more fancy programming footwork:

```
stems = [stemmer.stem(word) for word in tokens]
```

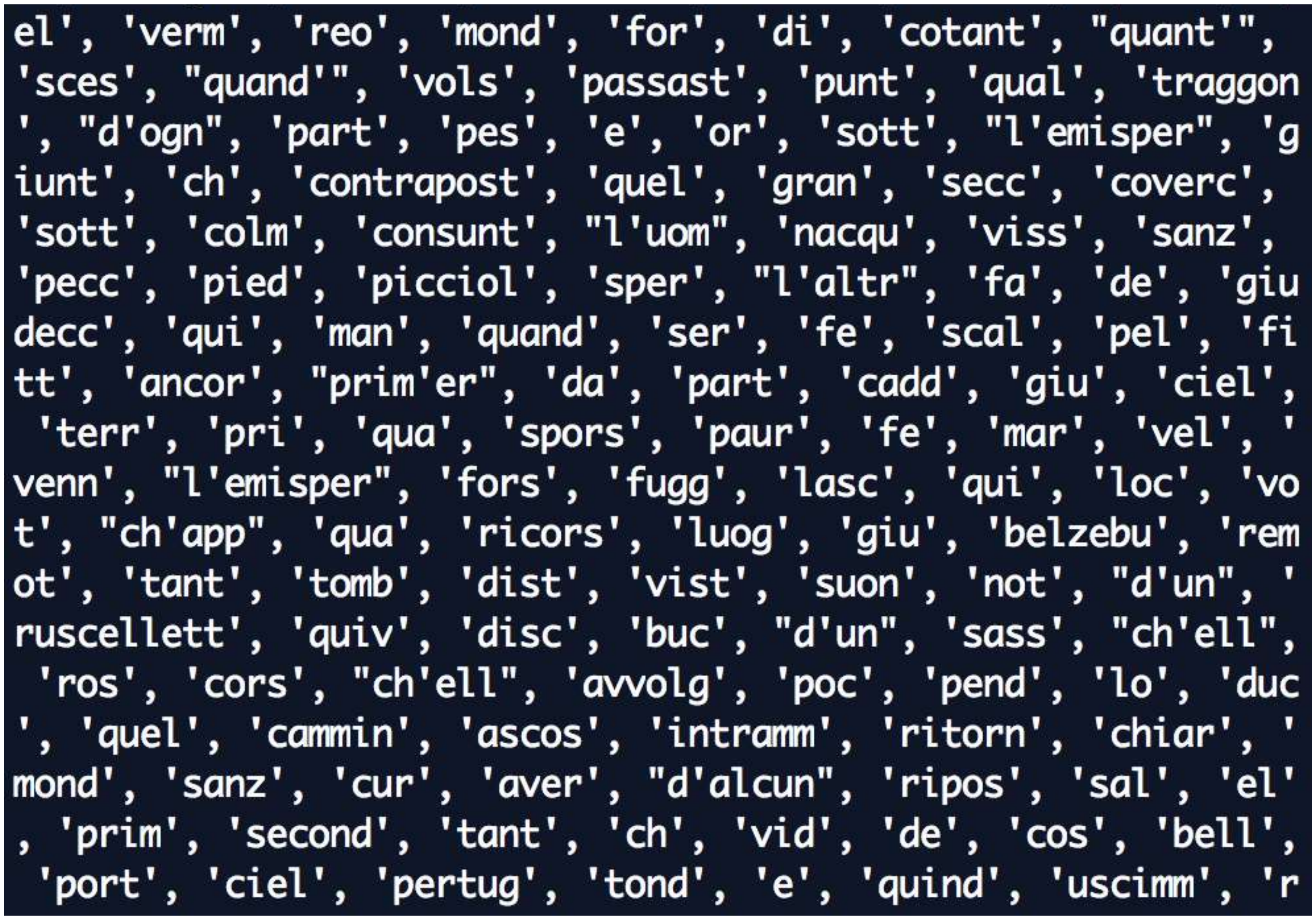
The line of code above basically translates to, "for every word (or token) in the `tokens` list, transform it into its stem, and store this new list of stems in a variable aptly called `stems`."

You can also print the stems with this line of code:

```
print(stems)
```



```
", 'divenn', 'allor', 'travagl', 'gent', 'gross', 'pens', 've
d', 'qual', 'quel', 'punt', "ch'i", 'ave', 'pass', 'lev', 'di
ss', 'maestr', 'pied', 'via', 'lung', 'cammin', 'malvag', 'gi
a', 'sol', 'mezz', 'terz', 'ried', 'non', 'cammin', 'palag',
"v'eravam", 'natural', 'burell', "ch'ave", 'mal', 'suol', 'lu
m', 'disag', 'prim', "ch'i", 'de', "l'abiss", 'divell', 'maes
tr', "diss'", 'quand', 'dritt', 'trarm', "d'err", 'poc', 'fav
ell', 'ov', 'ghiacc', 'com', 'fitt', 'sottosopr', "poc'or", '
ser', 'man', 'fatt', 'sol', 'tragitt', 'ed', 'elli', 'me', 't
u', 'imagin', 'ancor', "d'esser", 'centr', "ov'i", 'pres', 'p
```



```
el', 'verm', 'reo', 'mond', 'for', 'di', 'cotant', "quant'",
'sces', "quand'", 'vols', 'passast', 'punt', 'qual', 'traggon
', "d'ogn", 'part', 'pes', 'e', 'or', 'sott', "l'emisper", 'g
iunt', 'ch', 'contrapost', 'quel', 'gran', 'secc', 'coverc',
'sott', 'colm', 'consunt', "l'uom", 'nacqu', 'viss', 'sanz',
'pecc', 'pied', 'picciol', 'sper', "l'altr", 'fa', 'de', 'giu
decc', 'qui', 'man', 'quand', 'ser', 'fe', 'scal', 'pel', 'fi
tt', 'ancor', "prim'er", 'da', 'part', 'cadd', 'giu', 'ciel',
 'terr', 'pri', 'qua', 'spors', 'paur', 'fe', 'mar', 'vel', '
venn', "l'emisper", 'fors', 'fugg', 'lasc', 'qui', 'loc', 'vo
t', "ch'app", 'qua', 'ricors', 'luog', 'giu', 'belzebu', 'rem
ot', 'tant', 'tomb', 'dist', 'vist', 'suon', 'not', "d'un", '
ruscellett', 'quiv', 'disc', 'buc', "d'un", 'sass', "ch'ell",
 'ros', 'cors', "ch'ell", 'avvolg', 'poc', 'pend', 'lo', 'duc
', 'quel', 'cammin', 'ascos', 'intramm', 'ritorn', 'chiar', '
mond', 'sanz', 'cur', 'aver', "d'alcun", 'ripos', 'sal', 'el'
, 'prim', 'second', 'tant', 'ch', 'vid', 'de', 'cos', 'bell',
 'port', 'ciel', 'pertug', 'tond', 'e', 'quind', 'uscimm', 'r
```

Now that we've got our stems, we're ready for some frequency analysis!

Calculating word frequencies

Now for the part where we can attempt to analyze Dante's *Inferno* based on word frequencies! I titled this section "Calculating word frequencies," but the computer will be the only one doing math — we'll just be writing a few succinct lines of code. That's the beauty of NLTK!

To store a frequency distribution in a variable, we simply say:

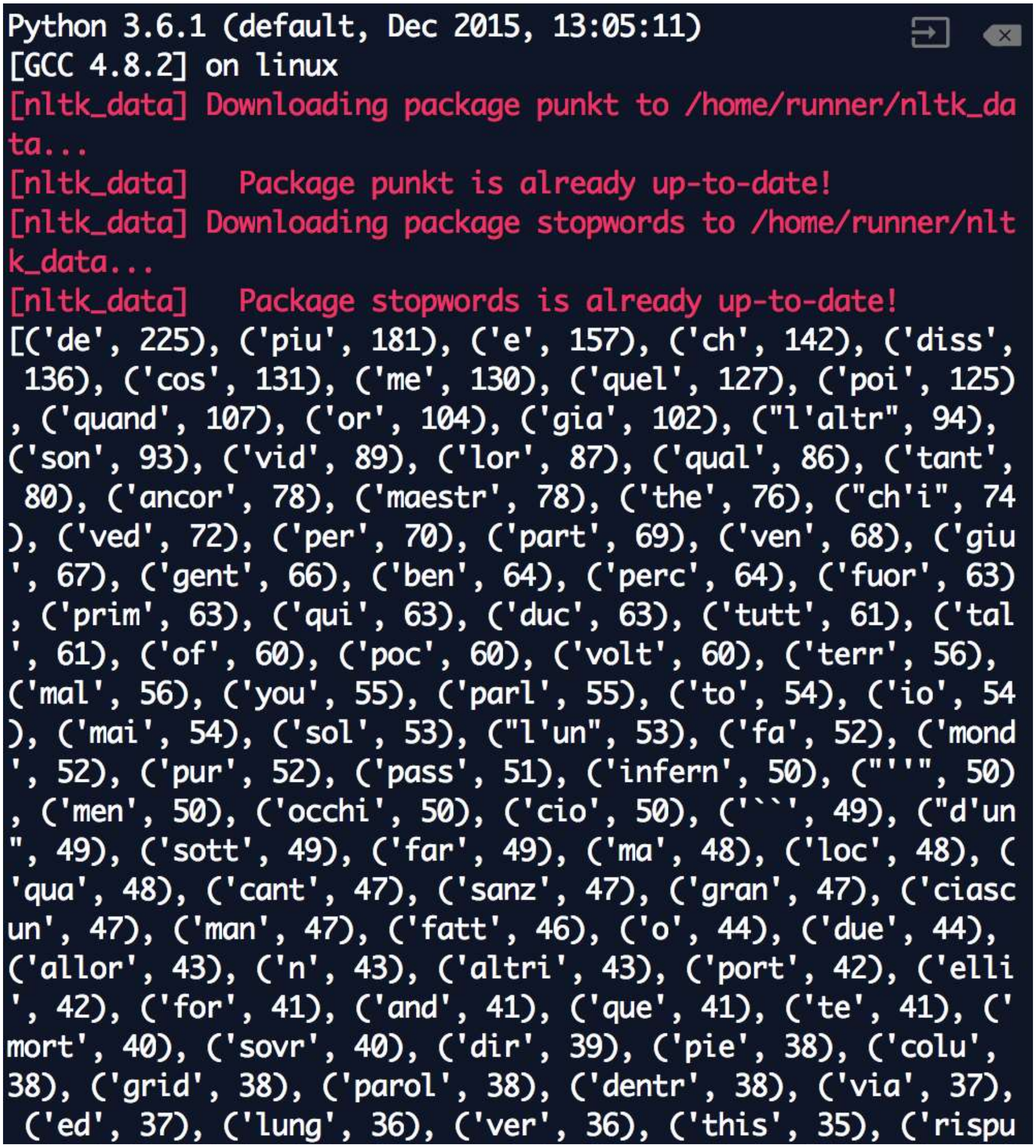
```
fdist = FreqDist(stems)
```

At this point, some programmers might do data visualizations, but if I'm not mistaken, repl.it doesn't have this capability — you'll have to download Python and the appropriate packages to your computer if you want to explore data viz.

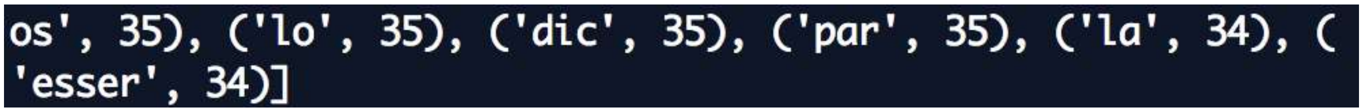
However, we can definitely print out values! With this line of code, I can print out the 100 most common words in Dante's *Inferno*:

```
print(fdist.most_common(100))
```

Cool, a list of the most common words!



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
[nltk_data] Downloading package punkt to /home/runner/nltk_da
ta...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/runner/nlt
k_data...
[nltk_data]   Package stopwords is already up-to-date!
[('de', 225), ('piu', 181), ('e', 157), ('ch', 142), ('diss',
 136), ('cos', 131), ('me', 130), ('quel', 127), ('poi', 125)
, ('quand', 107), ('or', 104), ('gia', 102), ("l'altr", 94),
('son', 93), ('vid', 89), ('lor', 87), ('qual', 86), ('tant',
 80), ('ancor', 78), ('maestr', 78), ('the', 76), ("ch'i", 74
), ('ved', 72), ('per', 70), ('part', 69), ('ven', 68), ('giu
', 67), ('gent', 66), ('ben', 64), ('perc', 64), ('fuor', 63)
, ('prim', 63), ('qui', 63), ('duc', 63), ('tutt', 61), ('tal
', 61), ('of', 60), ('poc', 60), ('volt', 60), ('terr', 56),
('mal', 56), ('you', 55), ('parl', 55), ('to', 54), ('io', 54
), ('mai', 54), ('sol', 53), ("l'un", 53), ('fa', 52), ('mond
', 52), ('pur', 52), ('pass', 51), ('infern', 50), ("''", 50)
, ('men', 50), ('occhi', 50), ('cio', 50), ('``', 49), ("d'un
", 49), ('sott', 49), ('far', 49), ('ma', 48), ('loc', 48), (
'qua', 48), ('cant', 47), ('sanz', 47), ('gran', 47), ('ciasc
un', 47), ('man', 47), ('fatt', 46), ('o', 44), ('due', 44),
('allor', 43), ('n', 43), ('altri', 43), ('port', 42), ('elli
', 42), ('for', 41), ('and', 41), ('que', 41), ('te', 41), ('
mort', 40), ('sovr', 40), ('dir', 39), ('pie', 38), ('colu',
38), ('grid', 38), ('parol', 38), ('dentr', 38), ('via', 37),
 ('ed', 37), ('lung', 36), ('ver', 36), ('this', 35), ('rispu
```



```
os', 35), ('lo', 35), ('dic', 35), ('par', 35), ('la', 34), (
'esser', 34)]
```

What conclusions can we draw by looking at word frequencies? Or, what questions are sparked by examining these frequencies?

We might notice that the word *piu* (more), is used 181 times, and the stem *tant*- (much), is used 80 times. These words suggest that hell is a place of extremes.

We might also notice that the stem *l'altr*- (the other), is used 94 times, which could lead to an investigation of duality in Dante's *Inferno*.

We could examine the most common verbs, which have to do with seeing and speaking, along with the word *occhi* (eyes), that appears 50 times. These words indicate the passivity with which Dante takes in the underworld.

Pretend you're a grad student in Italian literature. What else can you think of?