Examples for Portuguese Processing

This HOWTO contains a variety of examples relating to the Portuguese language. It is intended to be read in conjunction with the NLTK book (`http://www.nltk.org/book`). For instructions on running the Python interpreter, please see the section *Getting Started with Python*, in Chapter 1.

1 Python Programming, with Portuguese Examples

Chapter 1 of the NLTK book contains many elementary programming examples, all with English texts. In this section, we'll see some corresponding examples using Portuguese. Please refer to the chapter for full discussion. *Vamos!*

```
>>> from nltk.examples.pt import *
*** Introductory Examples for the NLTK Book ***
Loading ptext1, ... and psent1, ...
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
ptext1: Mem—rias P—stumas de Br‡s Cubas (1881)
ptext2: Dom Casmurro (1899)
ptext3: G%nesis
ptext4: Folha de Sau Paulo (1994)
>>>
```

Any time we want to find out about these texts, we just have to enter their names at the Python prompt:

```
>>> ptext2
<Text: Dom Casmurro (1899)>
```

1.1 Searching Text

A concordance permits us to see words in context.

```
>>> ptext1.concordance('olhos')
Building index...
Displaying 25 of 138 matches:
De pŽ , ^ cabeceira da cama , com os olhos estœpidos , a boca entreaberta , a t
orelhas . Pela minha parte fechei os olhos e deixei - me ir ^ ventura . J‡ agor
x›es de cŽrebro enfermo . Como ia de olhos fechados , n‹o via o caminho ; lembr
gelos eternos . Com efeito , abri os olhos e vi que o meu animal galopava numa
me apareceu ent‹o , fitando - me uns olhos rutilantes como o sol . Tudo nessa f
—mim mesmo . Ent‹o , encarei - a com olhos sœplices , e pedi mais alguns anos .
```

For a given word, we can find words with a similar text distribution:

```
>>> ptext1.similar('chegar')
acabada acudir aludir avistar bramanismo casamento cheguei com contar
contr‡rio corpo dali deixei desferirem dizer fazer filhos j‡ leitor lhe
>>> ptext3.similar('chegar')
achar alumiar arrombar destruir governar guardar ir lavrar passar que
toda tomar ver vir
```

We can search for the statistically significant collocations in a text:

```
>>> ptext1.collocations()
Building collocations list
Quincas Borba; Lobo Neves; alguma coisa; Br‡s Cubas; meu pai; dia
seguinte; n‹o sei; Meu pai; alguns instantes; outra vez; outra coisa;
por exemplo; mim mesmo; coisa nenhuma; mesma coisa; n‹o era; dias
depois; Passeio Pœblico; olhar para; das coisas
```

We can search for words in context, with the help of *regular expressions*, e.g.:

```
>>> ptext1.findall("<olhos> (<.*>)")
estœpidos; e; fechados; rutilantes; sœplices; a; do; babavam;
na; moles; se; da; umas; espraiavam; chamejantes; espetados;
```

We can automatically generate random text based on a given text, e.g.:

```
>>> ptext3.generate()
No princ'pio , criou Deus os aben□oou , dizendo : Onde { est‹o } e atŽ
^ ave dos cŽus , { que } ser‡ . Disse mais Abr‹o : D‡ - me a mulher
que tomaste ; porque daquele po□o Eseque , { tinha .} E disse : N‹o
poderemos descer ; mas , do campo ainda n‹o estava na casa do teu
pesco□o . E viveu Serugue , depois Sime‹o e Levi { s‹o } estes ? E o
var‹o , porque habitava na terra de Node , da m‹o de Esaœ : Jeœs ,
Jal‹o e Cor‡
```

1.2 Texts as List of Words

A few sentences have been defined for you.

```
>>> psent1
['o', 'amor', 'da', 'gl\xf3ria', 'era', 'a', 'coisa', 'mais',
'verdadeiramente', 'humana', 'que', 'h\xe1', 'no', 'homem', ',',
'e', ',', 'conseq\xfcentemente', ',', 'a', 'sua', 'mais',
'genu\xedna', 'fei\xe7\xe3o', '.']
>>>
```

Notice that the sentence has been *tokenized*. Each token is represented as a string, represented using quotes, e.g. `'coisa'`. Some strings contain special characters, e.g. `\xf3`, the internal representation for —. The tokens are combined in the form of a *list*. How long is this list?

```
>>> len(psent1)
25
>>>
```

What is the vocabulary of this sentence?

```
>>> sorted(set(psent1))
[',', '.', 'a', 'amor', 'coisa', 'conseq\xfcentemente', 'da', 'e'
'era', 'fei\xe7\xe3o', 'genu\xedna', 'gl\xf3ria', 'homem', 'humana',
'h\xe1', 'mais', 'no', 'o', 'que', 'sua', 'verdadeiramente']
>>>
```

Let's iterate over each item in `psent2`, and print information for each:

```
>>> for w in psent1:
...     print w.decode('latin-1'), len(w), w[-1]:
...
N‹o 3 o
consultes 9 s
dicion‡rios 11 s
. 1 .
```

Observe how we make a human-readable version of a string, using `decode()`. Also notice that we accessed the last character of a string `w` using `w[-1]`.

We just saw a `for` loop above. Another useful control structure is a *list comprehension*.

```
>>> [w.upper() for w in psent2]
['N\xc3O', 'CONSULTES', 'DICION\xc1RIOS', '.']
>>> [w for w in psent1 if w.endswith('a')]
['da', 'gl\xf3ria', 'era', 'a', 'coisa', 'humana', 'a', 'sua', 'genu\xedna']
>>> [w for w in ptext4 if len(w) > 15]
[u'norte-irlandeses', u'pan-nacionalismo', u'predominatemente', u'primeiro-ministro',
u'primeiro-ministro', u'irlandesa-americana', u'responsabilidades', u'significativamente']
```

We can examine the relative frequency of words in a text, using `FreqDist`:

```
>>> fd1 = FreqDist(ptext1)
>>> fd1
<FreqDist with 77098 outcomes>
>>> fd1['olhos']
137
>>> fd1.max()
u','
>>> fd1.samples()[:100]
[u',', u'.', u'a', u'que', u'de', u'e', u'-', u'o', u';', u'me', u'um', u'n\xe3o',
u'\x97', u'se', u'do', u'da', u'uma', u'com', u'os', u'\xe9', u'era', u'as', u'eu',
u'lhe', u'ao', u'em', u'para', u'mas', u'...', u'!', u'\xe0', u'na', u'mais', u'?',
u'no', u'como', u'por', u'N\xe3o', u'dos', u'ou', u'ele', u':', u'Virg\xedlia',
```

```
u'meu', u'disse', u'minha', u'das', u'O', u'/', u'A', u'CAP\xcdTULO', u'muito',
u'depois', u'coisa', u'foi', u'sem', u'olhos', u'ela', u'nos', u'tinha', u'nem',
u'E', u'outro', u'vida', u'nada', u'tempo', u'menos', u'outra', u'casa', u'homem',
u'porque', u'quando', u'mim', u'mesmo', u'ser', u'pouco', u'estava', u'dia',
u't\xe3o', u'tudo', u'Mas', u'at\xe9', u'D', u'ainda', u's\xf3', u'alguma',
u'la', u'vez', u'anos', u'h\xe1', u'Era', u'pai', u'esse', u'lo', u'dizer', u'assim',
u'ent\xe3o', u'dizia', u'aos', u'Borba']
```

2 Reading Corpora

2.1 Accessing the Machado Text Corpus

NLTK includes the complete works of Machado de Assis.

```
>>> from nltk.corpus import machado
>>> machado.fileids()
['contos/macn001.txt', 'contos/macn002.txt', 'contos/macn003.txt', ...]
```

Each file corresponds to one of the works of Machado de Assis. To see a complete list of works, you can look at the corpus README file: `print machado.readme()`. Let's access the text of the *Posthumous Memories of Br‡s Cubas*.

We can access the text as a list of characters, and access 200 characters starting from position 10,000.

```
>>> raw_text = machado.raw('romance/marm05.txt')
>>> raw_text[10000:10200]
u', primou no\nEstado, e foi um dos amigos particulares do vice-rei Conde
da Cunha.\n\nComo este apelido de Cubas lhe\ncheirasse excessivamente a
tanoaria, alegava meu pai, bisneto de Dami\xe3o, que o\ndito ape'
```

However, this is not a very useful way to work with a text. We generally think of a text as a sequence of words and punctuation, not characters:

```
>>> text1 = machado.words('romance/marm05.txt')
>>> text1
['Romance', ',', 'Mem\xf3rias', 'P\xf3stumas', 'de', ...]
>>> len(text1)
77098
>>> len(set(text1))
10848
```

Here's a program that finds the most common ngrams that contain a particular target word.

```
>>> from nltk import ingrams, FreqDist
>>> target_word = 'olhos'
>>> fd = FreqDist(ng
...               for ng in ingrams(text1, 5)
...               if target_word in ng)
>>> for hit in fd.samples():
...     print ' '.join(hit)
...
, com os olhos no
com os olhos no ar
com os olhos no ch‹o
e todos com os olhos
me estar com os olhos
os olhos est¤pidos , a
os olhos na costura ,
os olhos no ar ,
, com os olhos espetados
, com os olhos est¤pidos
, com os olhos fitos
, com os olhos naquele
, com os olhos para
```

2.2 Accessing the MacMorpho Tagged Corpus

NLTK includes the MAC-MORPHO Brazilian Portuguese POS-tagged news text, with over a million words of journalistic texts extracted from ten sections of the daily newspaper *Folha de Sao Paulo*, 1994.

We can access this corpus as a sequence of words or tagged words as follows:

```
>>> nltk.corpus.mac_morpho.words()
['Jersei', 'atinge', 'm\xe9dia', 'de', 'Cr$', '1,4', ...]
>>> nltk.corpus.mac_morpho.sents()
[['Jersei', 'atinge', 'm\xe9dia', 'de', 'Cr$', '1,4', 'milh\xe3o',
'em', 'a', 'venda', 'de', 'a', 'Pinhal', 'em', 'S\xe3o', 'Paulo'],
['Programe', 'sua', 'viagem', 'a', 'a', 'Exposi\xe7\xe3o', 'Nacional',
'do', 'Zebu', ',', 'que', 'come\xe7a', 'dia', '25'], ...]
>>> nltk.corpus.mac_morpho.tagged_words()
[('Jersei', 'N'), ('atinge', 'V'), ('m\xe9dia', 'N'), ...]
```

We can also access it in sentence chunks.

```
>>> nltk.corpus.mac_morpho.tagged_sents()
[[('Jersei', 'N'), ('atinge', 'V'), ('m\xe9dia', 'N'), ('de', 'PREP'),
  ('Cr$', 'CUR'), ('1,4', 'NUM'), ('milh\xe3o', 'N'), ('em', 'PREP|+'),
  ('a', 'ART'), ('venda', 'N'), ('de', 'PREP|+'), ('a', 'ART'),
  ('Pinhal', 'NPROP'), ('em', 'PREP'), ('S\xe3o', 'NPROP'),
  ('Paulo', 'NPROP')],
 [('Programe', 'V'), ('sua', 'PROADJ'), ('viagem', 'N'), ('a', 'PREP|+'),
  ('a', 'ART'), ('Exposi\xe7\xe3o', 'NPROP'), ('Nacional', 'NPROP'),
  ('do', 'NPROP'), ('Zebu', 'NPROP'), (',', ','), ('que', 'PRO-KS-REL'),
  ('come\xe7a', 'V'), ('dia', 'N'), ('25', 'N|AP')], ...]
```

This data can be used to train taggers (examples below for the Floresta treebank).

2.3 Accessing the Floresta Portuguese Treebank

The NLTK data distribution includes the "Floresta Sinta(c)tica Corpus" version 7.4, available from `http://www.linguateca.pt/Floresta/`.

We can access this corpus as a sequence of words or tagged words as follows:

```
>>> from nltk.corpus import floresta
>>> floresta.words()
['Um', 'revivalismo', 'refrescante', 'O', '7_e_Meio', ...]
>>> floresta.tagged_words()
[('Um', '>N+art'), ('revivalismo', 'H+n'), ...]
```

The tags consist of some syntactic information, followed by a plus sign, followed by a conventional part-of-speech tag. Let's strip off the material before the plus sign:

```
>>> def simplify_tag(t):
...     if "+" in t:
...         return t[t.index("+")+1:]
...     else:
...         return t
>>> twords = floresta.tagged_words()
>>> twords = [(w.lower(), simplify_tag(t)) for (w,t) in twords]
>>> twords[:10]
[('um', 'art'), ('revivalismo', 'n'), ('refrescante', 'adj'), ('o', 'art'), ('7_e_meio', 'prop'),
('\xe9', 'v-fin'), ('um', 'art'), ('ex-libris', 'n'), ('de', 'prp'), ('a', 'art')]
```

Pretty printing the tagged words:

```
>>> print ' '.join(word + '/' + tag for (word, tag) in twords[:10])
um/art revivalismo/n refrescante/adj o/art 7_e_meio/prop é/v-fin um/art ex-libris/n de/prp a/art
```

Count the word tokens and types, and determine the most common word:

```
>>> words = floresta.words()
>>> len(words)
211852
>>> fd = nltk.FreqDist(words)
>>> len(fd)
29421
>>> fd.max()
'de'
```

List the 20 most frequent tags, in order of decreasing frequency:

```
>>> tags = [simplify_tag(tag) for (word,tag) in floresta.tagged_words()]
>>> fd = nltk.FreqDist(tags)
>>> fd.keys()[:20]
['n', 'prp', 'art', 'v-fin', ',', 'prop', 'adj', 'adv', '.',
 'conj-c', 'v-inf', 'pron-det', 'v-pcp', 'num', 'pron-indp',
 'pron-pers', '\xab', '\xbb', 'conj-s', '}']
```

We can also access the corpus grouped by sentence:

```
>>> floresta.sents()
[['Um', 'revivalismo', 'refrescante'],
 ['O', '7_e_Meio', '\xe9', 'um', 'ex-libris', 'de', 'a', 'noite',
  'algarvia', '.'], ...]
>>> floresta.tagged_sents()
[[('Um', '>N+art'), ('revivalismo', 'H+n'), ('refrescante', 'N<+adj')],
 [('O', '>N+art'), ('7_e_Meio', 'H+prop'), ('\xe9', 'P+v-fin'),
  ('um', '>N+art'), ('ex-libris', 'H+n'), ('de', 'H+prp'),
  ('a', '>N+art'), ('noite', 'H+n'), ('algarvia', 'N<+adj'), ('.', '.')],
 ...]
>>> floresta.parsed_sents()
[Tree('UTT+np', [Tree('>N+art', ['Um']), Tree('H+n', ['revivalismo']),
                 Tree('N<+adj', ['refrescante'])]),
 Tree('STA+fcl',
     [Tree('SUBJ+np', [Tree('>N+art', ['O']),
                       Tree('H+prop', ['7_e_Meio'])]),
      Tree('P+v-fin', ['\xe9']),
      Tree('SC+np',
         [Tree('>N+art', ['um']),
          Tree('H+n', ['ex-libris']),
          Tree('N<+pp', [Tree('H+prp', ['de']),
                         Tree('P<+np', [Tree('>N+art', ['a']),
                                        Tree('H+n', ['noite']),
                                        Tree('N<+adj', ['algarvia'])])])]),
      Tree('.', ['.'])]), ...]
```

To view a parse tree, use the `draw()` method, e.g.:

```
>>> psents = floresta.parsed_sents()
>>> psents[5].draw()
```

2.4 Character Encodings

Python understands the common character encoding used for Portuguese, ISO 8859-1 (ISO Latin 1).

```
>>> import os, nltk.test
>>> testdir = os.path.split(nltk.test.__file__)[0]
>>> text = open(os.path.join(testdir, 'floresta.txt')).read()
>>> text[:60]
'O 7 e Meio \xe9 um ex-libris da noite algarvia.\n\xc9 uma das mais '
>>> print text[:60]
O 7 e Meio é um ex-libris da noite algarvia.
É uma das mais
>>> text[:60].decode('latin-1')
u'O 7 e Meio \xe9 um ex-libris da noite algarvia.\n\xc9 uma das mais '
>>> text[:60].decode('latin-1').encode('utf-16')
'\xff\xfeO\x00 \x007\x00 \x00e\x00 \x00M\x00e\x00i\x00o\x00 \x00\xe9\x00 \x00u\x00m\x00 \x00e\x00x\x00-\x001\x
```

For more information about character encodings and Python, please see section 3.3 of the book.

3 Processing Tasks

3.1 Simple Concordancing

Here's a function that takes a word and a specified amount of context (measured in characters), and generates a concordance for that word.

```
>>> def concordance(word, context=30):
...     for sent in floresta.sents():
...         if word in sent:
...             pos = sent.index(word)
...             left = ' '.join(sent[:pos])
...             right = ' '.join(sent[pos+1:])
```

```
...             print '%*s %s %-*s' %\
...                 (context, left[-context:], word, context, right[:context])
```

```
>>> concordance("dar")
anduru , foi o suficiente para dar a volta a o resultado .
             1. O P?BLICO veio dar a a imprensa di?ria portuguesa
  A fartura de pensamento pode dar maus resultados e n?s n?o quer
                      Come?a a dar resultados a pol?tica de a Uni
ial come?ar a incorporar- lo e dar forma a um ' site ' que tem se
r com Constantino para ele lhe dar tamb?m os pap?is assinados .
va a brincar , pois n?o lhe ia dar procura??o nenhuma enquanto n?
?rica como o ant?doto capaz de dar sentido a o seu enorme poder .
. . .
>>> concordance("vender")
er recebido uma encomenda para vender 4000 blindados a o Iraque .
m?rico_Amorim caso conseguisse vender o lote de ac??es de o empres?r
mpre ter jovens simp?ticos a ? vender ? chega ! }
       Disse que o governo vai vender ? desde autom?vel at? particip
ndiciou ontem duas pessoas por vender carro com ?gio .
        A inten??o de Fleury ? vender as a??es para equilibrar as fi
```

3.2 Part-of-Speech Tagging

Let's begin by getting the tagged sentence data, and simplifying the tags as described earlier.

```
>>> from nltk.corpus import floresta
>>> tsents = floresta.tagged_sents()
>>> tsents = [[(w.lower(),simplify_tag(t)) for (w,t) in sent] for sent in tsents if sent]
>>> train = tsents[100:]
>>> test = tsents[:100]
```

We already know that `n` is the most common tag, so we can set up a default tagger that tags every word as a noun, and see how well it does:

```
>>> tagger0 = nltk.DefaultTagger('n')
>>> nltk.tag.accuracy(tagger0, test)
0.17697228144989338
```

Evidently, about one in every six words is a noun. Let's improve on this by training a unigram tagger:

```
>>> tagger1 = nltk.UnigramTagger(train, backoff=tagger0)
>>> nltk.tag.accuracy(tagger1, test)
0.87029140014214645
```

Next a bigram tagger:

```
>>> tagger2 = nltk.BigramTagger(train, backoff=tagger1)
>>> nltk.tag.accuracy(tagger2, test)
0.89019189765458417
```

3.3 Sentence Segmentation

Punkt is a language-neutral sentence segmentation tool. We

```
>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/portuguese.pickle')
>>> raw_text = machado.raw('romance/marm05.txt')
>>> sentences = sent_tokenizer.tokenize(raw_text)
>>> for sent in sentences[1000:1005]:
...     print sent:
>>> for sent in sentences[1000:1005]:
...     print "<<", sent, ">>"
...
<< Em verdade, parecia ainda mais mulher do que era;
seria crian□a nos seus folgares de mo□a; mas assim quieta, impass'vel, tinha a
compostura da mulher casada. >>
<< Talvez essa circunst%ncia lhe diminu'a um pouco da
gra□a virginal. >>
<< Depressa nos familiarizamos; a m‹e fazia-lhe grandes elogios, eu
escutava-os de boa sombra, e ela sorria com os olhos fœlgidos, como se l‡ dentro
do cŽrebro lhe estivesse a voar uma borboletinha de asas de ouro e olhos de
diamante... >>
<< Digo l‡ dentro, porque c‡ fora o
```

```
que esvoaou foi uma borboleta preta, que subitamente penetrou na varanda, e
comeou a bater as asas em derredor de D. EusŽbia. >>
<< D. EusŽbia deu um grito,
levantou-se, praguejou umas palavras soltas: - T'esconjuro!... >>
```

The sentence tokenizer can be trained and evaluated on other text. The source text (from the Floresta Portuguese Treebank) contains one sentence per line. We read the text, split it into its lines, and then join these lines together using spaces. Now the information about sentence breaks has been discarded. We split this material into training and testing data:

```
>>> import os, nltk.test
>>> testdir = os.path.split(nltk.test.__file__)[0]
>>> text = open(os.path.join(testdir, 'floresta.txt')).read()
>>> lines = text.split('\n')
>>> train = ' '.join(lines[10:])
>>> test = ' '.join(lines[:10])
```

Now we train the sentence segmenter (or sentence tokenizer) and use it on our test sentences:

```
>>> stok = nltk.PunktSentenceTokenizer(train)
>>> print stok.tokenize(test)
['O 7 e Meio \xe9 um ex-libris da noite algarvia.',
'\xc9 uma das mais antigas discotecas do Algarve, situada em Albufeira,
que continua a manter os tra\xe7os decorativos e as clientelas de sempre.',
'\xc9 um pouco a vers\xe3o de uma esp\xe9cie de \xaboutro lado\xbb da noite,
a meio caminho entre os devaneios de uma fauna perif\xe9rica, seja de Lisboa,
Londres, Dublin ou Faro e Portim\xe3o, e a postura circunspecta dos fi\xe9is da casa,
que dela esperam a m\xfasica \xabgeracionista\xbb dos 60 ou dos 70.',
'N\xe3o deixa de ser, nos tempos que correm, um certo \xabvery typical\xbb algarvio,
cabe\xe7a de cartaz para os que querem fugir a algumas movimenta\xe7\xf5es nocturnas
j\xe1 a caminho da ritualiza\xe7\xe3o de massas, do g\xe9nero \xabvamos todos ao
Calypso e encontramo-nos na Locomia\xbb.',
'E assim, aos 2,5 milh\xf5es que o Minist\xe9rio do Planeamento e Administra\xe7\xe3o
do Territ\xf3rio j\xe1 gasta no pagamento do pessoal afecto a estes organismos,
v\xeam juntar-se os montantes das obras propriamente ditas, que os munic\xedpios,
j\xe1 com projectos na m\xe3o, v\xeam reivindicar junto do Executivo, como salienta
aquele membro do Governo.',
'E o dinheiro \xabn\xe3o falta s\xf3 \xe0s c\xe2maras\xbb, lembra o secret\xe1rio de Estado,
que considera que a solu\xe7\xe3o para as autarquias \xe9 \xabespecializarem-se em
fundos comunit\xe1rios\xbb.',
'Mas como, se muitas n\xe3o disp\xf5em, nos seus quadros, dos t\xe9cnicos necess\xe1rios?',
'\xabEncomendem-nos a projectistas de fora\xbb porque, se as obras vierem a ser financiadas,
eles at\xe9 saem de gra\xe7a, j\xe1 que, nesse caso, \xabos fundos comunit\xe1rios pagam
os projectos, o mesmo n\xe3o acontecendo quando eles s\xe3o feitos pelos GAT\xbb,
dado serem organismos do Estado.',
'Essa poder\xe1 vir a ser uma hip\xf3tese, at\xe9 porque, no terreno, a capacidade dos GAT
est\xe1 cada vez mais enfraquecida.',
'Alguns at\xe9 j\xe1 desapareceram, como o de Castro Verde, e outros t\xeam vindo a perder quadros.']
```

NLTK's data collection includes a trained model for Portuguese sentence segmentation, which can be loaded as follows. It is faster to load a trained model than to retrain it.

```
>>> stok = nltk.data.load('tokenizers/punkt/portuguese.pickle')
```

3.4 Stemming

NLTK includes the RSLP Portuguese stemmer. Here we use it to stem some Portuguese text:

```
>>> stemmer = nltk.stem.RSLPStemmer()
>>> stemmer.stem("copiar")
u'copi'
>>> stemmer.stem("paisagem")
u'pais'
```

3.5 Stopwords

NLTK includes Portuguese stopwords:

```
>>> stopwords = nltk.corpus.stopwords.words('portuguese')
>>> stopwords[:10]
['a', 'ao', 'aos', 'aquela', 'aquelas', 'aquele', 'aqueles', 'aquilo', 'as', 'at\xe9']
```

Now we can use these to filter text. Let's find the most frequent words (other than stopwords) and print them in descending order of frequency:

```
>>> fd = nltk.FreqDist(w.lower() for w in floresta.words() if w not in stopwords)
>>> for word in fd.keys()[:20]:
...     print word, fd[word]
, 13444
. 7725
« 2369
» 2310
é 1305
o 1086
} 1047
{ 1044
a 897
; 633
em 516
ser 466
sobre 349
os 313
anos 301
ontem 292
ainda 279
segundo 256
ter 249
dois 231
```