pronouncing Documentation

Release 0.2.0

Allison Parrish

Jul 02, 2018

Contents

Pronouncing is a simple interface for the CMU Pronouncing Dictionary. The library is designed to be easy to use, and has no external dependencies. For example, here's all you need to do in order to find rhymes for a given word:

```
>>> import pronouncing
>>> pronouncing.rhymes("climbing")
['diming', 'liming', 'priming', 'rhyming', 'timing']
```

Read the documentation here: https://pronouncing.readthedocs.org.

I made this library because I wanted to be able to use the CMU Pronouncing Dictionary in my projects without having to install the grand behemoth that is NLTK. It's designed to be friendly to beginner programmers who want to get started with creative language generation and analysis, and for experts who want to make quick prototypes of projects that deal with English pronunciation.

CHAPTER 1

Installation

Install with pip like so:

```
pip install pronouncing
```

You can also download the source code and install manually:

```
python setup.py install
```

CHAPTER 2

Contents

2.1 Tutorial and Cookbook

This tutorial will demonstrate how to perform several common tasks with the Pronouncing library and provide a few examples of how the library can be used creatively.

2.1.1 Word pronunciations

Let's start by using Pronouncing to get the pronunciation for a given word. Here's the code:

```
>>> import pronouncing
>>> pronouncing.phones_for_word("permit")
[u'P ER0 M IH1 T', u'P ER1 M IH2 T']
```

The *pronouncing.phones_for_word()* function returns a list of all pronunciations for the given word found in the CMU pronouncing dictionary. Pronunciations are given using a special phonetic alphabet known as ARPAbet. Here's a list of ARPAbet symbols and what English sounds they stand for. Each token in a pronunciation string is called a "phone." The numbers after the vowels indicate the vowel's stress. The number `1` indicates primary stress; `2` indicates secondary stress; and `0` indicates unstressed. (Wikipedia has a good overview of how stress works in English, if you're interested.)

Sometimes, the pronouncing dictionary has more than one pronunciation for the same word. "Permit" is a good example: it can be pronounced either with the stress on the first syllable ("do you have a permit to program here?") or on the second syllable ("will you permit me to program here?"). For this reason, the *pronouncing.phones_for_word()* function returns a list of possible pronunciations. (You'll need to come up with your own criteria for deciding which pronunciation is best for your purposes.)

Here's how to calculate the most common sounds in a given text:

```
>>> import pronouncing
>>> from collections import Counter
>>> text = "april is the cruelest month breeding lilacs out of the dead"
>>> count = Counter()
```

(continues on next page)

(continued from previous page)

```
>>> words = text.split()
>>> for word in words:
...   pronunciation_list = pronouncing.phones_for_word(word)
...   if len(pronunciation_list) > 0:
...     count.update(pronunciation_list[0].split(" "))
...
>>> count.most_common(5)
[(u'AH0', 4), (u'L', 4), (u'D', 3), (u'R', 3), (u'DH', 2)]
```

2.1.2 Pronunciation search

Pronouncing has a helpful function *pronouncing.search()* which allows you to search the pronouncing dictionary for words whose pronunciation matches a particular regular expression. For example, to find words that have within them the same sounds as the word "sighs":

```
>>> import pronouncing
>>> phones = pronouncing.phones_for_word("sighs")[0]
>>> pronouncing.search(phones)[:5]
[u'incise', u'incised', u'incisor', u'incisors', u'malloseismic']
```

For convenience, word-boundary anchors (`\b`) are added automatically to the beginning and end of the pattern you pass to *pronouncing.search()*. You're free to include any other regular expression syntax in the pattern. Here's another example, which finds all of the words that end in "-iddle":

```
>>> pronouncing.search("IH1 D AH0 L$")[:5]
[u'biddle', u'criddle', u'fiddle', u'friddle', u'kiddle']
```

Another example, which re-writes a text by taking each word and replacing it with a random word that begins with the same first two phones:

```
>>> import pronouncing
>>> import random
>>> text = 'april is the cruelest month breeding lilacs out of the dead'
>>> out = list()
>>> for word in text.split():
...   phones = pronouncing.phones_for_word(word)[0]
...   first2 = phones.split()[:2]
...   out.append(random.choice(pronouncing.search("^" + " ".join(first2))))
...
>>> print ' '.join(out)
apec's isn't them kraatz muffy bronte leichliter outpacing of than delfs
```

2.1.3 Counting syllables

To get the number of syllables in a word, first get one of its pronunciations with *pronouncing.phones_for_word()* and pass the resulting string of phones to the *pronouncing.syllable_count()* function, like so:

```
>>> import pronouncing
>>> pronunciation_list = pronouncing.phones_for_word("programming")
>>> pronouncing.syllable_count(pronunciation_list[0])
3
```

The following example calculates the total number of syllables in a text (assuming that all of the words are found in the pronouncing dictionary):

```
>>> import pronouncing
>>> text = "april is the cruelest month breeding lilacs out of the dead"
>>> phones = [pronouncing.phones_for_word(p)[0] for p in text.split()]
>>> sum([pronouncing.syllable_count(p) for p in phones])
15
```

2.1.4 Meter

Pronouncing includes a number of functions to help you isolate metrical characteristics of a text. You can use the *pronouncing.stresses()* function to get a string that represents the "stress pattern" of a string of phones:

```
>>> import pronouncing
>>> phones_list = pronouncing.phones_for_word("snappiest")
>>> pronouncing.stresses(phones_list[0])
u'102'
```

A "stress pattern" is a string that contains only the stress values from a sequence of phones. (The numbers indicate the level of stress: `1` for primary stress, `2` for secondary stress, and `0` for unstressed.)

You can use the *pronouncing.search_stresses()* function to find words based on their stress patterns. For example, to find words that have two dactyls in them ("dactyl" is a metrical foot consisting of one stressed syllable followed by two unstressed syllables):

```
>>> import pronouncing
>>> pronouncing.search_stresses("100100")
[u'afroamerican', u'afroamericans', u'interrelationship', u'overcapacity']
```

You can use regular expression syntax inside of the patterns you give to *pronouncing.search_stresses()*. For example, to find all words wholly consisting of two anapests (unstressed, unstressed, stressed), with "stressed" meaning either primary stress or secondary stress:

```
>>> import pronouncing
>>> pronouncing.search_stresses("^00[12]00[12]$")
[u'neopositivist', u'undercapitalize', u'undercapitalized']
```

The following example rewrites a text, replacing each word with a random word that has the same stress pattern:

```
>>> import pronouncing
>>> import random
>>> text = 'april is the cruelest month breeding lilacs out of the dead'
>>> for word in text.split():
...   pronunciations = pronouncing.phones_for_word(word)
...   pat = pronouncing.stresses(pronunciations[0])
...   replacement = random.choice(pronouncing.search_stresses("^"+pat+"$"))
...   out.append(replacement)
...
>>> ' '.join(out)
u"joneses kopf whats rathbun p's gavan midpoint nill goh the pont's"
```

2.1.5 Rhyme

Pronouncing includes a simple function, *pronouncing.rhymes()*, which returns a list of words that (potentially) rhyme with a given word. You can use it like so:

```
>>> import pronouncing
>>> pronouncing.rhymes("failings")
[u'mailings', u'railings', u'tailings']
```

The *pronouncing.rhymes()* function returns a list of all possible rhymes for the given word—i.e., words that rhyme with any of the given word's pronunciations. If you only want rhymes for one particular pronunciation, the the *pronouncing.rhyming_part()* function gives a smaller part of a string of phones that can be used with *pronouncing.search()* to find rhyming words. The following code demonstrates how to find rhyming words for two different pronunciations of "uses":

```
>>> import pronouncing
>>> pronunciations = pronouncing.phones_for_word("uses")
>>> sss = pronouncing.rhyming_part(pronunciations[0])
>>> zzz = pronouncing.rhyming_part(pronunciations[1])
>>> pronouncing.search(sss + "$")[:5]
[u"bruce's", u'juices', u'medusas', u'produces', u"tuscaloosa's"]
>>> pronouncing.search(zzz + "$")[:5]
[u'abuses', u'cabooses', u'disabuses', u'excuses', u'induces']
```

Use the `in` operator to check to see if one word rhymes with another:

```
>>> import pronouncing
>>> "wheeze" in pronouncing.rhymes("cheese")
True
>>> "geese" in pronouncing.rhymes("cheese")
False
```

The following example rewrites a text, replacing each word with a rhyming word (when a rhyming word is available):

```
>>> import pronouncing
>>> import random
>>> text = 'april is the cruelest month breeding lilacs out of the dead'
>>> out = list()
>>> for word in text.split():
...   rhymes = pronouncing.rhymes(word)
...   if len(rhymes) > 0:
...     out.append(random.choice(rhymes))
...   else:
...     out.append(word)
...
>>> print ' '.join(out)
april wiles's duh coolest month ceding pontiac's krout what've worthey wehde
```

2.1.6 Next steps

Hopefully this is just the beginning of your rhyme- and meter-filled journey. Consult *Pronouncing API Reference* for more information about individual functions in the library.

Pronouncing is just one possible interface for the CMU pronouncing dictionary, and you may find that for your particular purposes, a more specialized approach is necessary. In that case, feel free to peruse Pronouncing's source code for helpful hints and tidbits.

2.2 Pronouncing API Reference

pronouncing.**init_cmu**(*filehandle=None*)
Initialize the module's pronunciation data.

This function is called automatically the first time you attempt to use another function in the library that requires loading the pronunciation data from disk. You can call this function manually to control when and how the pronunciation data is loaded (e.g., you're using this module in a web application and want to load the data asynchronously).

Parameters **filehandle** – a filehandle with CMUdict-formatted data

Returns None

pronouncing.**parse_cmu**(*cmufh*)
Parses an incoming file handle as a CMU pronouncing dictionary file.

(Most end-users of this module won't need to call this function explicitly, as it's called internally by the *init_cmu()* function.)

Parameters **cmufh** – a filehandle with CMUdict-formatted data

Returns a list of 2-tuples pairing a word with its phones (as a string)

pronouncing.**phones_for_word**(*find*)
Get the CMUdict phones for a given word.

Because a given word might have more than one pronunciation in the dictionary, this function returns a list of all possible pronunciations.

```
>>> import pronouncing
>>> pronouncing.phones_for_word("permit")
['P ER0 M IH1 T', 'P ER1 M IH2 T']
```

Parameters **find** – a word to find in CMUdict.

Returns a list of phone strings that correspond to that word.

pronouncing.**rhymes**(*word*)
Get words rhyming with a given word.

This function may return an empty list if no rhyming words are found in the dictionary, or if the word you pass to the function is itself not found in the dictionary.

```
>>> import pronouncing
>>> pronouncing.rhymes("conditioner")
['commissioner', 'parishioner', 'petitioner', 'practitioner']
```

Parameters **word** – a word

Returns a list of rhyming words

pronouncing.**rhyming_part**(*phones*)
Get the "rhyming part" of a string with CMUdict phones.

"Rhyming part" here means everything from the vowel in the stressed syllable nearest the end of the word up to the end of the word.

```
>>> import pronouncing
>>> phones = pronouncing.phones_for_word("purple")
>>> pronouncing.rhyming_part(phones[0])
'ER1 P AH0 L'
```

Parameters **phones** – a string containing space-separated CMUdict phones

Returns a string with just the "rhyming part" of those phones

pronouncing.**search**(*pattern*)

Get words whose pronunciation matches a regular expression.

This function Searches the CMU dictionary for pronunciations matching a given regular expression. (Word boundary anchors are automatically added before and after the pattern.)

```
>>> import pronouncing
>>> 'interpolate' in pronouncing.search('ER1 P AH0')
True
```

Parameters **pattern** – a string containing a regular expression

Returns a list of matching words

pronouncing.**search_stresses**(*pattern*)

Get words whose stress pattern matches a regular expression.

This function is a special case of *search()* that searches only the stress patterns of each pronunciation in the dictionary. You can get stress patterns for a word using the *stresses_for_word()* function.

```
>>> import pronouncing
>>> pronouncing.search_stresses('020120')
['gubernatorial']
```

Parameters **pattern** – a string containing a regular expression

Returns a list of matching words

pronouncing.**stresses**(*s*)

Get the vowel stresses for a given string of CMUdict phones.

Returns only the vowel stresses (i.e., digits) for a given phone string.

```
>>> import pronouncing
>>> pronouncing.stresses(pronouncing.phones_for_word('obsequious')[0])
'0100'
```

Parameters **s** – a string of CMUdict phones

Returns string of just the stresses

pronouncing.**stresses_for_word**(*find*)

Get a list of possible stress patterns for a given word.

```
>>> import pronouncing
>>> pronouncing.stresses_for_word('permit')
['01', '12']
```

Parameters **find** – a word to find

Returns a list of possible stress patterns for the given word.

pronouncing.**syllable_count**(*phones*)

Count the number of syllables in a string of phones.

To find the number of syllables in a word, call *phones_for_word()* first to get the CMUdict phones for that word.

```
>>> import pronouncing
>>> phones = pronouncing.phones_for_word("literally")
>>> pronouncing.syllable_count(phones[0])
4
```

Parameters **phones** – a string containing space-separated CMUdict phones

Returns integer count of syllables in list of phones

2.3 Credits and Acknowledgements

Lead developer: Allison Parrish <allison@decontextualize.com>.

This package was originally developed as part of my Spring 2015 research fellowship at ITP. Thank you to the program and its students for their interest and support!

2.4 History

2.4.1 0.2.0 (2018-07-01)

- Removed dictionary data from this package in favor of a dependency on David L. Day's very nice cmudict package.
- Many fixes and improvements from hugovk (thanks!)

2.4.2 0.1.5 (2017-04-13)

- Messed up the PyPI upload. Yay!

2.4.3 0.1.4 (2017-04-12)

- Improved performance when retrieving rhyming words. (Based on pull request proposed by WillPiledriver.)

2.4.4 0.1.3 (2017-01-17)

- Various tweaks and performance improvements.

2.4.5 0.1.2 (2015-06-23)

- Pre-compiled regex for improved performance. (Contributed by John Wiseman.)

2.4.6 0.1.1 (2015-06-12)

- First release on PyPI.

Python Module Index

p

Index