

---

# **jellyfish Documentation**

*Release 0.5.6*

**James Turk**

December 01, 2016



<b>1</b>	<b>Overview</b>	<b>1</b>
1.1	Phonetic Encoding . . . . .	1
1.1.1	American Soundex . . . . .	1
1.1.2	Metaphone . . . . .	1
1.1.3	NYSIIS . . . . .	1
1.1.4	Match Rating Approach (codex) . . . . .	2
1.2	Stemming . . . . .	2
1.2.1	Porter Stemmer . . . . .	2
1.3	String Comparison . . . . .	2
1.3.1	Levenshtein Distance . . . . .	2
1.3.2	Damerau-Levenshtein Distance . . . . .	3
1.3.3	Hamming Distance . . . . .	3
1.3.4	Jaro Distance . . . . .	3
1.3.5	Jaro-Winkler Distance . . . . .	3
1.3.6	Match Rating Approach (comparison) . . . . .	3
1.4	Changelog . . . . .	4
1.4.1	0.5.6 - June 23 2016 . . . . .	4
1.4.2	0.5.5 - June 21 2016 . . . . .	4
1.4.3	0.5.4 - May 13 2016 . . . . .	4
1.4.4	0.5.3 - March 15 2016 . . . . .	4
1.4.5	0.5.2 - February 3 2016 . . . . .	4
1.4.6	0.5.1 - July 12 2015 . . . . .	4
1.4.7	0.5.0 - April 23 2015 . . . . .	4
1.4.8	0.4.0 - March 27 2015 . . . . .	4
1.4.9	0.3.4 - February 4 2015 . . . . .	5
1.4.10	0.3.3 - November 20 2014 . . . . .	5
1.4.11	0.3.2 - August 11 2014 . . . . .	5
1.4.12	0.3.1 - July 16 2014 . . . . .	5
1.4.13	0.3.0 - July 15 2014 . . . . .	5
1.4.14	0.2.2 - March 14 2014 . . . . .	5
1.4.15	0.2.0 - January 26 2012 . . . . .	5
1.4.16	0.1.2 - September 16 2010 . . . . .	5
<b>2</b>	<b>Implementation</b>	<b>7</b>
<b>3</b>	<b>Indices and tables</b>	<b>9</b>



jellyfish is a library of functions for approximate and phonetic matching of strings.

The library provides implementations of the following algorithms:

## 1.1 Phonetic Encoding

These algorithms convert a string to a normalized phonetic encoding, converting a word to a representation of its pronunciation. Each takes a single string and returns a coded representation.

### 1.1.1 American Soundex

**soundex** (*s*)

Calculate the American Soundex of the string *s*.

Soundex is an algorithm to convert a word (typically a name) to a four digit code in the form 'A123' where 'A' is the first letter of the name and the digits represent similar sounds.

For example `soundex('Ann') == soundex('Anne') == 'A500'` and `soundex('Rupert') == soundex('Robert') == 'R163'`.

See the [Soundex article at Wikipedia](#) for more details.

### 1.1.2 Metaphone

**metaphone** (*s*)

Calculate the metaphone code for the string *s*.

The metaphone algorithm was designed as an improvement on Soundex. It transforms a word into a string consisting of '0BFHJKLMNPRSTWXY' where '0' is pronounced 'th' and 'X' is a '[sc]h' sound.

For example `metaphone('Klumpz') == metaphone('Clumps') == 'KLMPX'`.

See the [Metaphone article at Wikipedia](#) for more details.

### 1.1.3 NYSIIS

**nysiis** (*s*)

Calculate the NYSIIS code for the string *s*.

The NYSIIS algorithm is an algorithm developed by the New York State Identification and Intelligence System. It transforms a word into a phonetic code. Like soundex and metaphone it is primarily intended for use on names (as they would be pronounced in English).

For example `nysiis('John') == nysiis('Jan') == JAN`.

See the [NYSIIS article at Wikipedia](#) for more details.

### 1.1.4 Match Rating Approach (codex)

**match\_rating\_codex**(*s*)

Calculate the match rating approach value (also called PNI) for the string *s*.

The Match rating approach algorithm is an algorithm for determining whether or not two names are pronounced similarly. The algorithm consists of an encoding function (similar to soundex or nysiis) which is implemented here as well as `match_rating_comparison()` which does the actual comparison.

See the [Match Rating Approach article at Wikipedia](#) for more details.

## 1.2 Stemming

### 1.2.1 Porter Stemmer

**porter\_stem**(*s*)

Reduce the string *s* to its stem using the common Porter stemmer.

Stemming is the process of reducing a word to its root form, for example 'stemmed' to 'stem'.

Martin Porter's algorithm is a common algorithm used for stemming that works for many purposes.

See the [official homepage for the Porter Stemming Algorithm](#) for more details.

## 1.3 String Comparison

These methods are all measures of the difference (aka *edit distance*) between two strings.

### 1.3.1 Levenshtein Distance

**levenshtein\_distance**(*s1*, *s2*)

Compute the Levenshtein distance between *s1* and *s2*.

Levenshtein distance represents the number of insertions, deletions, and substitutions required to change one word to another.

For example: `levenshtein_distance('berne', 'born') == 2` representing the transformation of the first e to o and the deletion of the second e.

See the [Levenshtein distance article at Wikipedia](#) for more details.

### 1.3.2 Damerau-Levenshtein Distance

**damerau\_levenshtein\_distance** (*s1*, *s2*)

Compute the Damerau-Levenshtein distance between *s1* and *s2*.

A modification of Levenshtein distance, Damerau-Levenshtein distance counts transpositions (such as ifhs for fish) as a single edit.

Where `levenshtein_distance('fish', 'ifsh') == 2` as it would require a deletion and an insertion, though `damerau_levenshtein_distance('fish', 'ifsh') == 1` as this counts as a transposition.

See the [Damerau-Levenshtein distance article at Wikipedia](#) for more details.

### 1.3.3 Hamming Distance

**hamming\_distance** (*s1*, *s2*)

Compute the Hamming distance between *s1* and *s2*.

Hamming distance is the measure of the number of characters that differ between two strings.

Typically Hamming distance is undefined when strings are of different length, but this implementation considers extra characters as differing. For example `hamming_distance('abc', 'abcd') == 1`.

See the [Hamming distance article at Wikipedia](#) for more details.

### 1.3.4 Jaro Distance

**jaro\_distance** (*s1*, *s2*)

Compute the Jaro distance between *s1* and *s2*.

Jaro distance is a string-edit distance that gives a floating point response in [0,1] where 0 represents two completely dissimilar strings and 1 represents identical strings.

### 1.3.5 Jaro-Winkler Distance

**jaro\_winkler** (*s1*, *s2*)

Compute the Jaro-Winkler distance between *s1* and *s2*.

Jaro-Winkler is a modification/improvement to Jaro distance, like Jaro it gives a floating point response in [0,1] where 0 represents two completely dissimilar strings and 1 represents identical strings.

See the [Jaro-Winkler distance article at Wikipedia](#) for more details.

### 1.3.6 Match Rating Approach (comparison)

**match\_rating\_comparison** (*s1*, *s2*)

Compare *s1* and *s2* using the match rating approach algorithm, returns `True` if strings are considered equivalent or `False` if not. Can also return `None` if *s1* and *s2* are not comparable (length differs by more than 3).

The Match rating approach algorithm is an algorithm for determining whether or not two names are pronounced similarly. Strings are first encoded using `match_rating_codex()` then compared according to the MRA algorithm.

See the [Match Rating Approach article at Wikipedia](#) for more details.

## 1.4 Changelog

### 1.4.1 0.5.6 - June 23 2016

- bugfix for metaphone & soundex raising unexpected TypeErrors on Windows (#54)

### 1.4.2 0.5.5 - June 21 2016

- bugfix for metaphone WH case

### 1.4.3 0.5.4 - May 13 2016

- bugfix for C version of damerau\_levenshtein thanks to Tyler Sellon

### 1.4.4 0.5.3 - March 15 2016

- style/packaging changes

### 1.4.5 0.5.2 - February 3 2016

- testing fixes for Python 3.5
- bugfix for Metaphone w/ silent H thanks to Jeremy Carbaugh

### 1.4.6 0.5.1 - July 12 2015

- bugfixes for NYSIIS
- bugfixes for metaphone
- bugfix for C version of jaro\_winkler

### 1.4.7 0.5.0 - April 23 2015

- consistent unicode behavior, all functions take unicode and reject bytes on Py2 and 3, C and Python
- parametrize tests
- Windows compiler support

### 1.4.8 0.4.0 - March 27 2015

- tons of new tests
- documentation
- split out cjellyfish
- test all w/ unicode and plenty of fixes to accommodate
- 100% test coverage



#### **1.4.9 0.3.4 - February 4 2015**

- fix segfaults and memory leaks via Danrich Parrol

#### **1.4.10 0.3.3 - November 20 2014**

- fix bugs in damerau and NYSIIS

#### **1.4.11 0.3.2 - August 11 2014**

- fix for jaro-winkler from David McKean
- more packaging fixes

#### **1.4.12 0.3.1 - July 16 2014**

- packaging fix for C/Python alternative

#### **1.4.13 0.3.0 - July 15 2014**

- python alternatives where C isn't available

#### **1.4.14 0.2.2 - March 14 2014**

- testing fixes
- assorted bugfixes in NYSIIS

#### **1.4.15 0.2.0 - January 26 2012**

- incorporate some speed changes from Peter Scott
- segfault bugfixes.

#### **1.4.16 0.1.2 - September 16 2010**

- initial working release



---

## Implementation

---

Each algorithm has C and Python implementations.

On a typical CPython install the C implementation will be used. The Python versions are available for PyPy and systems where compiling the CPython extension is not possible.



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## D

damerau\_levenshtein\_distance() (built-in function), 3

## H

hamming\_distance() (built-in function), 3

## J

jaro\_distance() (built-in function), 3

jaro\_winkler() (built-in function), 3

## L

levenshtein\_distance() (built-in function), 2

## M

match\_rating\_codex() (built-in function), 2

match\_rating\_comparison() (built-in function), 3

metaphone() (built-in function), 1

## N

nysiis() (built-in function), 1

## P

porter\_stem() (built-in function), 2

## S

soundex() (built-in function), 1