Dragon Mapper Documentation

Release 0.2.6

Thomas Roten

May 23, 2016

Contents

Dragon Mapper is a Python library that provides identification and conversion functions for Chinese text processing:

- Identify a string as Traditional or Simplified Chinese, Pinyin, or Zhuyin.
- Convert between Chinese characters, Pinyin, Zhuyin, and the International Phonetic Alphabet.

```
>>> s = ''
>>> dragonmapper.hanzi.is_simplified(s)
True
>>> dragonmapper.hanzi.to_pinyin(s)
'wshìygèměiguórén'
>>> dragonmapper.hanzi.to_pinyin(s, all_readings=True)
'[w][shì/shi/tí][y][gè/ge/gě/gàn][měi][guó][rén/ren]'

>>> s = 'W shì ygè měiguórén.'
>>> dragonmapper.transcriptions.is_pinyin(s)
True
>>> dragonmapper.transcriptions.pinyin_to_zhuyin(s)
'ˇ    ˇ  .'
>>> dragonmapper.transcriptions.pinyin_to_ipa(s)
'w  i k me kw n.'
```

If this is your first time using Dragon Mapper, check out the *Installation*. Then, read the *Tutorial*.

If you want a more in-depth view of Dragon Mapper, check out the *API*.

If you're looking to help out, read *Contributing*.

CHAPTER 1

Support

If you encounter a bug, have a feature request, or need help using Dragon Mapper, then use Dragon Mapper's GitHub Issues page to send feedback.

CHAPTER 2

Documentation Contents

2.1 Dragon Mapper

Dragon Mapper is a Python library that provides identification and conversion functions for Chinese text processing.

- Documentation: http://dragonmapper.rtfd.org
- GitHub: https://github.com/tsroten/dragonmapper
- Free software: MIT license

2.1.1 Features

- Convert between Chinese characters, Pinyin, Zhuyin, and the International Phonetic Alphabet.
- Identify a string as Traditional or Simplified Chinese, Pinyin, Zhuyin, or the International Phonetic Alphabet.

```
>>> s = ''
>>> dragonmapper.hanzi.is_simplified(s)
True
>>> dragonmapper.hanzi.to_pinyin(s)
'wshìygèměiguórén'
>>> dragonmapper.hanzi.to_pinyin(s, all_readings=True)
'[w][shì/shi/tí][y][gè/ge/gě/gàn][měi][guó][rén/ren]'

>>> s = 'W shì ygè měiguórén.'
>>> dragonmapper.transcriptions.is_pinyin(s)
True
>>> dragonmapper.transcriptions.pinyin_to_zhuyin(s)
'ˇ    ˇ  .'
>>> dragonmapper.transcriptions.pinyin_to_ipa(s)
'w  i k me kw n.'
```

2.1.2 Getting Started

- Install Dragon Mapper
- Read Dragon Mapper's tutorial
- Report bugs and ask questions via GitHub Issues
- Refer to the API documentation when you need more technical information

- Contribute documentation, code, or feedback

2.2 Installation

Installing Dragon Mapper is easy. Make sure you have Python 2.7 or 3 along with Zhon and Hanzi Identifier. Then use pip:

```
$ pip install dragonmapper
```

That will download Dragon Mapper from the Python Package Index and install it in your Python's `site-packages` directory.

2.2.1 Tarball Release

If you'd rather install Dragon Mapper manually:

1. Download the most recent release from Dragon Mapper's PyPi page.
2. Unpack the tarball.
3. From inside the directory `dragonmapper-XX`, run `python setup.py install`

That will install Dragon Mapper in your Python's `site-packages` directory.

2.2.2 Install the Development Version

Dragon Mapper's code is hosted at GitHub. To install the development version first make sure Git is installed. Then run:

```
$ git clone git://github.com/tsroten/dragonmapper.git
$ pip install -e dragonmapper
```

This will link the `dragonmapper` directory into your `site-packages` directory.

2.2.3 Running the Tests

Running the tests is easy:

```
$ python setup.py test
```

If you want to run the tests using multiple versions of Python, install and run tox:

```
$ pip install tox
$ tox
```

Dragon Mapper's `tox.ini` file is configured to run tests using Python 2.7, 3.3, and 3.4. It will also build the documentation (requires Sphinx).

2.3 Tutorial

This tutorial will walk you through common tasks involving Dragon Mapper and its two supported data formats: Chinese characters and Chinese transcriptions. Not all of Dragon Mapper's functions or their options are explained here. Be sure to read the *API* for further information.

Note: Python 2 strings are not Unicode by default. Prefix the strings in these code samples with 'u' to make them work correctly. For example, `u''` instead of `''`. See Unicode Literals in Python Source Code for more information.

2.3.1 Working with Chinese Characters

When using Dragon Mapper to work with Chinese characters, you will first want to import Dragon Mapper's `dragonmapper.hanzi` module:

```
>>> from dragonmapper import hanzi
```

It will take a second or two for Dragon Mapper to load the CC-CEDICT and Unihan data into memory.

Convert Characters to Readings

Let's take a look at a common task: converting a string of Chinese characters to Pinyin. We'll be using the function `dragonmapper.hanzi.to_pinyin()`.

```
>>> s = ''
>>> hanzi.to_pinyin(s)
'zhègèzìzěnmeniàn'
```

As you can see, Dragon Mapper simply replaced each Chinese character with it's most common reading. Dragon Mapper will automatically add apostrophes to separate syllables if needed. That is all you need for simple cases. However, you may want to include all possible readings just in case the most common reading is incorrect.

```
>>> hanzi.to_pinyin(s, all_readings=True)
'[zhè][gè/ge/gě/gàn][zì/zi][zěn][me/yo/mó/ma][niàn]'
```

In the previous examples, Dragon Mapper converted each character separately. Most of the time, you will want to segment your text into words and convert whole words instead of just characters. Just separate the words by spaces or Chinese punctuation marks and Dragon Mapper will recognize the word boundaries.

```
>>> # Sentence without word boundaries marked.
... s = ''
>>> hanzi.to_pinyin(s)
'zhègèhěnbiànyi'

>>> # Sentence with word boundaries marked.
... s_spaced = '  '
>>> hanzi.to_pinyin(s_spaced)
'zhège hěn piànyi'

>>> hanzi.to_pinyin(s_spaced, all_readings=True)
'[zhège] [hěn] [piànyi/biànyí]'
```

Dragon Mapper's `dragonmapper.hanzi.to_zhuyin()` and `dragonmapper.hanzi.to_ipa()` work just like the above examples.

Identifying Chinese Characters

Identifying a string of Chinese as containing Traditional versus Simplified characters is a difficult task that involves a lot more than merely looking at each character on its own. That task is best left up to humans. However, it can also be helpful to get a general idea of what character system a string is compatible with. Dragon Mapper can assist with that.

dragonmapper.hanzi.identify() and its related functions can identify Chinese characters as Traditional or Simplified based on the CC-CEDICT dictionary. Again, don't see this as a fool proof way to determine a string's identity. Instead, look at it as a way to determine what character system a string is compatible with. Let's take a look:

```
>>> s = ''
>>> hanzi.identify(s) is hanzi.SIMPLIFIED
True

>>> # Shortcut functions are provided:
... hanzi.is_simplified(s)
True
>>> hanzi.is_traditional(s)
False
```

The Traditional and Simplified Chinese character systems share some characters. Sometimes a string can be compatible with both character systems:

```
>>> s = ''
>>> hanzi.identify(s) is hanzi.BOTH
True

>>> # Using the shortcut functions:
... hanzi.is_traditional(s)
True
>>> hanzi.is_simplified(s)
True
```

Sometimes, a string might contain characters that exist exclusively in Traditional Chinese and characters that exist exclusively in Simplified:

```
>>> s = '. . .'
>>> hanzi.identify(s) is hanzi.MIXED
True

>>> hanzi.has_chinese(s)
True
>>> # It's not compatible with Traditional or Simplified Chinese:
... hanzi.is_traditional(s)
False
>>> hanzi.is_simplified(s)
False
```

The last scenario is a string that doesn't contain any Chinese characters:

```
>>> s = 'Hello. My name is Thomas.'
>>> hanzi.identify(s) is hanzi.UNKNOWN
True

>>> hanzi.has_chinese(s)
False
```

2.3.2 Working with Transcriptions

When using Dragon Mapper to work with Chinese transcriptions, you will first want to import Dragon Mapper's dragonmapper.transcriptions module:

```
>>> from dragonmapper import transcriptions
```

Identifying Transcription Systems

Dragon Mapper supports three transcription systems: Pinyin (accented and numbered), Zhuyin (Bopomofo), and the International Phonetic Alphabet (IPA).

Let's try to identify which transcription system a string is:

```
>>> s = 'W shì ygè měiguórén.'
>>> transcriptions.identify(s) is transcriptions.PINYIN
True

>>> # Shortcut functions:
... transcriptions.is_pinyin(s)
True
>>> transcriptions.is_zhuyin(s)
False
>>> transcriptions.is_ipa(s)
False

>>> s = 'ˇ ˇ'
>>> transcriptions.identify(s) is transcriptions.ZHUYIN
True

>>> # Shortcut functions:
... transcriptions.is_zhuyin(s)
True
>>> transcriptions.is_pinyin(s)
False
>>> transcriptions.is_ipa(s)
False
```

The functions above operate on a syllable-level to check whether or not a Pinyin or Zhuyin string is valid. However, this can take awhile, so if you don't need to validate a string on the syllable-level, consider validating it on a character-level with `is_pinyin_compatible()` or `is_zhuyin_compatible()`

```
>>> s = 'W shì ygè měiguórén.'
>>> transcriptions.is_pinyin_compatible(s)
True
```

Converting Transcription Systems

Converting between Pinyin, Zhuyin, and IPA is simple. The syllables have a one-to-one correspondence. Let's see how Dragon Mapper handles it:

```
>>> zhuyin = 'ˇ ˇ'
>>> pinyin = transcriptions.zhuyin_to_pinyin(zhuyin)
>>> ipa = transcriptions.zhuyin_to_ipa(zhuyin)

>>> print(pinyin)
n ho
>>> print(ipa)
ni x
```

Pinyin apostrophes are handled automatically when converting to/from Pinyin. If you're into using middle dots for tone markers, those are supported as well.

If you have a string and you don't know what transcription system it's using, but you know what system you want to convert it to, Dragon Mapper has some handy functions to help you:

```
>>> unknown = 'nho'
>>> transcriptions.to_zhuyin(unknown)
'ˇ ˇ'

>>> # If it's already in the target transcription, no conversion is done.
... transcriptions.to_pinyin(unknown)
'nho'
```

`dragonmapper.transcriptions.to_pinyin()`, `dragonmapper.transcriptions.to_zhuyin()`, and `dragonmapper.transcriptions.to_ipa()` all work like that.

2.3.3 Conclusion

You've seen that Dragon Mapper understands two data formats: Chinese characters and Chinese transcriptions. Dragon Mapper has both identification and conversion capabilities.

Not all of Dragon Mapper's functions or their options were explained above. Be sure to read the *API* for further information.

2.4 API

2.4.1 dragonmapper.hanzi

Identification and transcription functions for Chinese characters.

Importing this module takes a moment because it loads CC-CEDICT and Unihan data into memory.

Identifying Chinese Characters

Identifying a string of text as Traditional or Simplified Chinese is a complicated task. This module takes a simple approach that only looks at individual characters and not word choice. When these functions identify a string of text as Simplified, they aren't saying, "This string of Chinese *is* Simplified Chinese and *not* Traditional Chinese." Instead, see it as identifying the string as *compatible* with the Simplified Chinese character system.

Note: These identification functions and constants are imported from the Hanzi Identifier library.

The following constants are used as return values for `identify()`.

`dragonmapper.hanzi.`**`UNKNOWN`**
: Indicates that a string doesn't contain any Chinese characters.

`dragonmapper.hanzi.`**`TRAD`**
`dragonmapper.hanzi.`**`TRADITIONAL`**
: Indicates that a string contains Chinese characters that are only used in Traditional Chinese.

`dragonmapper.hanzi.`**`SIMP`**
`dragonmapper.hanzi.`**`SIMPLIFIED`**
: Indicates that a string contains Chinese characters that are only used in Simplified Chinese.

`dragonmapper.hanzi.`**`BOTH`**
: Indicates that a string contains Chinese characters that are compatible with both Traditional and Simplified Chinese.

dragonmapper.hanzi.**MIXED**
Indicates that a string contains Chinese characters that are found exclusively in Traditional and Simplified Chinese.

dragonmapper.hanzi.**identify**()
Identify what kind of Chinese characters a string contains.

s is a string to examine. The string's Chinese characters are tested to see if they are compatible with the Traditional or Simplified characters systems, compatible with both, or contain a mixture of Traditional and Simplified characters. The TRADITIONAL, SIMPLIFIED, BOTH, or MIXED constants are returned to indicate the string's identity. If *s* contains no Chinese characters, then UNKNOWN is returned.

All characters in a string that aren't found in the CC-CEDICT dictionary are ignored.

Because the Traditional and Simplified Chinese character systems overlap, a string containing Simplified characters could identify as SIMPLIFIED or BOTH depending on if the characters are also Traditional characters. To make testing the identity of a string easier, the functions is_traditional(), is_simplified(), and has_chinese() are provided.

dragonmapper.hanzi.**has_chinese**()
Check if a string has Chinese characters in it.

This is a faster version of:

```
>>> identify('foo') is not UNKNOWN
```

dragonmapper.hanzi.**is_traditional**()
Check if a string's Chinese characters are Traditional.

This is equivalent to:

```
>>> identify('foo') in (TRADITIONAL, BOTH)
```

dragonmapper.hanzi.**is_simplified**()
Check if a string's Chinese characters are Simplified.

This is equivalent to:

```
>>> identify('foo') in (SIMPLIFIED, BOTH)
```

Transcribing Chinese Characters

The following functions transliterate Chinese characters into various transcription systems.

dragonmapper.hanzi.**to_pinyin**(*s*, *delimiter=' '*, *all_readings=False*, *container='[]'*, *accented=True*)
Convert a string's Chinese characters to Pinyin readings.

s is a string containing Chinese characters. *accented* is a boolean value indicating whether to return accented or numbered Pinyin readings.

delimiter is the character used to indicate word boundaries in *s*. This is used to differentiate between words and characters so that a more accurate reading can be returned.

all_readings is a boolean value indicating whether or not to return all possible readings in the case of words/characters that have multiple readings. *container* is a two character string that is used to enclose words/characters if *all_readings* is True. The default '[]' is used like this: '[READING1/READING2]'.

Characters not recognized as Chinese are left untouched.

dragonmapper.hanzi.**to_zhuyin**(*s*, *delimiter=' '*, *all_readings=False*, *container='[]'*)
Convert a string's Chinese characters to Zhuyin readings.

s is a string containing Chinese characters.

delimiter is the character used to indicate word boundaries in *s*. This is used to differentiate between words and characters so that a more accurate reading can be returned.

all_readings is a boolean value indicating whether or not to return all possible readings in the case of words/characters that have multiple readings. *container* is a two character string that is used to enclose words/characters if *all_readings* is `True`. The default `'[]'` is used like this: `'[READING1/READING2]'`.

Characters not recognized as Chinese are left untouched.

dragonmapper.hanzi.**to_ipa**(*s*, *delimiter=' '*, *all_readings=False*, *container='[]'*)
Convert a string's Chinese characters to IPA.

s is a string containing Chinese characters.

delimiter is the character used to indicate word boundaries in *s*. This is used to differentiate between words and characters so that a more accurate reading can be returned.

all_readings is a boolean value indicating whether or not to return all possible readings in the case of words/characters that have multiple readings. *container* is a two character string that is used to enclose words/characters if *all_readings* is `True`. The default `'[]'` is used like this: `'[READING1/READING2]'`.

Characters not recognized as Chinese are left untouched.

2.4.2 dragonmapper.transcriptions

Identification and conversion functions for Chinese transcription systems.

Identifying Chinese Transcriptions

The following constants are used as return values for `identify()`.

dragonmapper.transcriptions.**UNKNOWN**
Indicates that a string isn't a recognized Chinese transcription.

dragonmapper.transcriptions.**PINYIN**
Indicates that a string's content consists of Pinyin.

dragonmapper.transcriptions.**ZHUYIN**
Indicates that a string's content consists of Zhuyin (Bopomofo).

dragonmapper.transcriptions.**IPA**
Indicates that a string's content consists of the International Phonetic Alphabet (IPA).

dragonmapper.transcriptions.**identify**(*s*)
Identify a given string's transcription system.

s is the string to identify. The string is checked to see if its contents are valid Pinyin, Zhuyin, or IPA. The `PINYIN`, `ZHUYIN`, and `IPA` constants are returned to indicate the string's identity. If *s* is not a valid transcription system, then `UNKNOWN` is returned.

When checking for valid Pinyin or Zhuyin, testing is done on a syllable level, not a character level. For example, just because a string is composed of characters used in Pinyin, doesn't mean that it will identify as Pinyin; it must actually consist of valid Pinyin syllables. The same applies for Zhuyin.

When checking for IPA, testing is only done on a character level. In other words, a string just needs to consist of Chinese IPA characters in order to identify as IPA.

The following functions use `identify()`, but don't require typing the names of the module-level constants.

dragonmapper.transcriptions.**is_pinyin**(*s*)
Check if *s* consists of valid Pinyin.

dragonmapper.transcriptions.**is_zhuyin**(*s*)
Check if *s* consists of valid Zhuyin.

dragonmapper.transcriptions.**is_ipa**(*s*)
Check if *s* consists of valid Chinese IPA.

The above functions `is_pinyin()` and `is_zhuyin()` check for valid syllables. This takes more time than checking on the character-level, but is more accurate. If you want to simply know if a string is compatible with Pinyin or Zhuyin, but don't need to know if each syllable is actually valid, then use these functions:

dragonmapper.transcriptions.**is_pinyin_compatible**(*s*)
Checks if *s* is consists of Pinyin-compatible characters.

This does not check if *s* contains valid Pinyin syllables; for that see `is_pinyin()`.

This function checks that all characters in *s* exist in `zhon.pinyin.printable`.

dragonmapper.transcriptions.**is_zhuyin_compatible**(*s*)
Checks if *s* is consists of Zhuyin-compatible characters.

This does not check if *s* contains valid Zhuyin syllables; for that see `is_zhuyin()`.

Besides Zhuyin characters and tone marks, spaces are also accepted. This function checks that all characters in *s* exist in `zhon.zhuyin.characters`, `zhon.zhuyin.marks`, or `' '`.

Converting Chinese Transcriptions

Converting between the various transcription systems is fairly simple. A few things to note:

- When converting from Pinyin to Zhuyin or IPA, spaces are added between each syllable because Zhuyin and IPA are not meant to be read in sentence format. They don't have the equivalent of Pinyin's apostrophe to separate certain syllables.
- When converting from Pinyin to Zhuyin or IPA, all syllable-separating apostrophes are removed. Those that don't separate syllables (like quotation marks) are left untouched.
- In Pinyin, `'v'` is considered another way to write `'ü'`. The `*_to_pinyin` functions all output that vowel as `'ü'`.

These conversion functions come in two flavors: functions that convert individual syllabes and functions that convert sentence-style text. If you only have individual syllables to convert, it's quicker to use the `*_syllable_to_*` functions that assume the input is a single valid syllable.

Syllable Conversion

dragonmapper.transcriptions.**numbered_syllable_to_accented**(*s*)
Convert numbered Pinyin syllable *s* to an accented Pinyin syllable.

It implements the following algorithm to determine where to place tone marks:

1. **If the syllable has an 'a', 'e', or 'o' (in that order), put the** tone mark over that vowel.
2. Otherwise, put the tone mark on the last vowel.

dragonmapper.transcriptions.**accented_syllable_to_numbered**(*s*)
Convert accented Pinyin syllable *s* to a numbered Pinyin syllable.

dragonmapper.transcriptions.**pinyin_syllable_to_zhuyin**(*s*)
Convert Pinyin syllable *s* to a Zhuyin syllable.

dragonmapper.transcriptions.**pinyin_syllable_to_ipa**(*s*)
Convert Pinyin syllable *s* to an IPA syllable.

dragonmapper.transcriptions.**zhuyin_syllable_to_pinyin**(*s*, *accented=True*)
Convert Zhuyin syllable *s* to a Pinyin syllable.

If *accented* is `True`, diacritics are added to the Pinyin syllable. If it's `False`, numbers are used to indicate the syllable's tone.

dragonmapper.transcriptions.**zhuyin_syllable_to_ipa**(*s*)
Convert Zhuyin syllable *s* to an IPA syllable.

dragonmapper.transcriptions.**ipa_syllable_to_pinyin**(*s*, *accented=True*)
Convert IPA syllable *s* to a Pinyin syllable.

If *accented* is `True`, diacritics are added to the Pinyin syllable. If it's `False`, numbers are used to indicate the syllable's tone.

dragonmapper.transcriptions.**ipa_syllable_to_zhuyin**(*s*)
Convert IPA syllable *s* to a Zhuyin syllable.

Sentence-Style Conversion

dragonmapper.transcriptions.**numbered_to_accented**(*s*)
Convert all numbered Pinyin syllables in *s* to accented Pinyin.

dragonmapper.transcriptions.**accented_to_numbered**(*s*)
Convert all accented Pinyin syllables in *s* to numbered Pinyin.

dragonmapper.transcriptions.**pinyin_to_zhuyin**(*s*)
Convert all Pinyin syllables in *s* to Zhuyin.

Spaces are added between connected syllables and syllable-separating apostrophes are removed.

dragonmapper.transcriptions.**pinyin_to_ipa**(*s*)
Convert all Pinyin syllables in *s* to IPA.

Spaces are added between connected syllables and syllable-separating apostrophes are removed.

dragonmapper.transcriptions.**zhuyin_to_pinyin**(*s*, *accented=True*)
Convert all Zhuyin syllables in *s* to Pinyin.

If *accented* is `True`, diacritics are added to the Pinyin syllables. If it's `False`, numbers are used to indicate tone.

dragonmapper.transcriptions.**zhuyin_to_ipa**(*s*)
Convert all Zhuyin syllables in *s* to IPA.

dragonmapper.transcriptions.**ipa_to_pinyin**(*s*, *accented=True*)
Convert all IPA syllables in *s* to Pinyin.

If *accented* is `True`, diacritics are added to the Pinyin syllables. If it's `False`, numbers are used to indicate tone.

dragonmapper.transcriptions.**ipa_to_zhuyin**(*s*)
Convert all IPA syllables in *s* to Zhuyin.

Combined: Identification and Conversion

These functions take an unidentified transcription string and identify it, then convert it into the target transcription system. If you know you'll be identifying your strings before you convert them, these can save you a few lines of code.

`dragonmapper.transcriptions.`**`to_pinyin`**`(`*s*, *accented=True*`)`
Convert *s* to Pinyin.

If *accented* is `True`, diacritics are added to the Pinyin syllables. If it's `False`, numbers are used to indicate tone.

`dragonmapper.transcriptions.`**`to_zhuyin`**`(`*s*`)`
Convert *s* to Zhuyin.

`dragonmapper.transcriptions.`**`to_ipa`**`(`*s*`)`
Convert *s* to IPA.

2.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/tsroten/dragonmapper/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Dragon Mapper could always use more documentation, whether as part of the official Dragon Mapper docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tsroten/dragonmapper/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.5.2 Get Started!

Ready to contribute? Here's how to set up *dragonmapper* for local development.

1. Fork the *dragonmapper* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/dragonmapper.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv dragonmapper
   $ cd dragonmapper/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 dragonmapper tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

2.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.7/3.3 and for PyPy. Check https://travis-ci.org/tsroten/dragonmapper/pull_requests and make sure that the tests pass for all supported Python versions.
4. If you want to receive credit, add your name to *AUTHORS.rst*.

2.6 Credits

2.6.1 Author and Maintainer

- Thomas Roten <https://github.com/tsroten>

2.6.2 Contributors

None yet. Why not be the first?

2.7 Change Log

2.7.1 0.2.6 (2016-05-23)

- Fixes reading for . Fixes #10.
- Add a note about Unicode string for Python 2 users.
- Bumps required hanzidentifier version.
- Fix umlaut on "l" consonant. Fixes #14.

2.7.2 0.2.5 (2015-08-06)

- Fixes #9. Uses io.open() in setup.py with UTF-8 encoding.

2.7.3 0.2.4 (2015-04-08)

- Fixes #8. Adds re.UNICODE to transcription conversion.
- Fixes misformatted readings for certain characters.
- Fixes #7. Fixes incorrect Unihan Database readings for the 'ou' vowel combinations.

2.7.4 0.2.3 (2014-04-28)

- Fixes #6. Adds -r suffix syllable to transcription mapping data.

2.7.5 0.2.2 (2014-04-28)

- Fixes a capitalization bug related to #5.

2.7.6 0.2.1 (2014-04-28)

- Reformats `README.rst`.
- Renames change log file to `*.rst`.
- Adds authors and contributing files.
- Sets up Travis CI.
- Adds version to `__init__.py`.
- Fixes #5. Make `accented_to_numbered()` add apostrophes when needed.
- Fixes #4. Fixes `numbered_to_accented()` handling of `'v'` vowel.
- Fixes #3. Changes `IndexError` exception handlers to `KeyError`.
- Fixes #2. Fixes `accented_to_numbered()` with uppercase accented vowel.

2.7.7 0.2.0 (2014-04-14)

- Fixes typo in is_pinyin.
- Adds is_pinyin_compatible() and is_zhuyin_compatible() functions.
- Removes code for identifying Hanzi and incorporates Hanzi Identifier library.
- Removes Sphinx viewcode extension.
- Adds Python 3.4 environment to tox configuration.
- Fixes typo in setup.py. Fixes #1.

2.7.8 0.1.0 (2014-02-17)

- Initial release.

d