# PyNLPIR Documentation

***Release 0.5.2***

**Thomas Roten**

March 25, 2017

Contents

PyNLPIR is a Python wrapper around the NLPIR/ICTCLAS Chinese segmentation software.

PyNLPIR allows you to easily segment Chinese text using NLPIR, one of the most widely-regarded Chinese text analyzers:

```
import pynlpir
pynlpir.open()

s = 'NLPIR'
pynlpir.segment(s)

[('', 'verb'), ('', 'noun'), ('', 'noun'), ('', 'punctuation mark'), ('', 'noun'), ('', 'noun'), ('',
```

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

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

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

CHAPTER 1

# Support

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

CHAPTER 2

# Documentation Contents

## Installation

Installing PyNLPIR is simple. PyNLPIR is designed to run on Python 2.7 or 3. Because of the included NLPIR library files, it only runs on Windows, GNU/Linux, or macOS.

### Pip

Install PyNLPIR using pip:

```
$ pip install pynlpir
$ pynlpir update
```

This will download PyNLPIR from the Python Package Index and install it in your Python's `site-packages` directory. `pynlpir update` will download the latest license from the NLPIR project.

### Install from Source

If you'd rather install PyNLPIR manually:

1. Download the most recent release from PyNLPIR's PyPi page.
2. Unpack the archive file.
3. From inside the directory `PyNLPIR-XX`, run `python setup.py install`
4. Run `pynlpir update` to download the latest license file.

This will install PyNLPIR in your Python's `site-packages` directory.

### Install the Development Version

PyNLPIR'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/pynlpir.git
$ pip install -e pynlpir
$ pynlpir update
```

This will link the `PyNLPIR` directory into your `site-packages` directory. `pynlpir update` will download the latest license from the NLPIR project.

### Running the Tests

Running the tests is easy. After downloading and unpacking PyNLPIR's source, run the following code from inside PyNLPIR's source directory:

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

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

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

## Tutorial

Now that you have *PyNLPIR installed*, let's look at how to use it.

There are two ways to use PyNLPIR: directly using the `ctypes` interface provided by PyNLPIR or using PyNLPIR's helper functions. The `ctypes` interface is more extensive, but more rigid. The helper functions are easy-to-use, but don't provide access to every NLPIR function. You can also use a mixture of the two methods. First, let's look at the helper functions.

### PyNLPIR Helper Functions

The helper functions are located in PyNLPIR's `__init__.py` file, so they are accessible by importing `pynlpir` directly.

#### Initializing NLPIR

Importing PyNLPIR loads the NLPIR API library automatically:

```
import pynlpir
```

Once it's imported, call `open()` to tell NLPIR to open the data files and initialize the API. See `open()`'s documentation for information on specifying a different data directory.

```
pynlpir.open()
```

By default, input is assumed to be unicode or UTF-8 encoded. If you'd like to use a different encoding (e.g. GBK or BIG5), use the *encoding* keyword argument when calling `open()`:

```
pynlpir.open(encoding='big5')
```

**Tip:** No matter what encoding you've specified, you can always pass unicode strings to `pynlpir` functions.

PyNLPIR's helper functions always return unicode strings.

Once you've initialized NLPIR, you can start segmenting and analyzing text.

#### Segmenting Text

Let's segment a lengthy sentence:

```
s = 'NLPIR2000ICTCLAS2009NLPIRNLPIR'
pynlpir.segment(s)

# Sample output: [('NLPIR', 'noun'), ('', 'verb'), ('', 'noun'), ('', 'noun'), ('', 'preposition'),
```

If you don't want part of speech tagging, call `segment()` with *pos_tagging* set to `False`:

```
pynlpir.segment(s, pos_tagging=False)

# Sample output: ['NLPIR', '', '', '', '', '2000', '', . . . ]
```

You can also customize how the part of speech tags are shown. By default, only the most generic part of speech name is used, i.e. the parent (for example, `'noun'` instead of `'transcribed toponym'`). If you'd like the most specific part of speech name instead, i.e. the child, set *pos_names* to `'child'`:

```
pynlpir.segment(s, pos_names='child')
```

If you want even more information about the part of speech tags, you can set *pos_names* to `'all'` and a part of speech hierarchy is returned (for example, `'noun:toponym:transcribed toponym'`):

```
pynlpir.segment(s, pos_names='all')
```

By default, part of speech tags are returned in English. If you'd like to see Chinese instead (e.g. `''` instead of `'noun'`), set *pos_english* to `False`:

```
pynlpir.segment(s, pos_english=False)
```

### Getting Key Words

Another useful function is `get_key_words()`:

```
pynlpir.get_key_words(s, weighted=True)
[('NLPIR', 2.08), ('', 1.74)]
```

`get_key_words()` analyzes the given Chinese text string and returns words that NLPIR considers key words. If *weighted* is `True`, then the key word's weight is also returned as a `float`.

### Closing the API

Now that we've looked at a brief introduction to PyNLPIR's helper functions, let's look at how to close the API.

When you're done using PyNLPIR, you can free up allocated memory by calling `close()`:

```
pynlpir.close()
```

## `ctypes` NLPIR Interface

`pynlpir.nlpir` provides access to NLPIR's C functions via `ctypes`. You can call them directly without bothering with the helper functions above. These functions work almost exactly the same as their C counterparts.

`pynlpir.nlpir` includes the module-level constants exported by NLPIR that are needed for calling many of its functions (e.g. encoding and part of speech constants). See the API page on `pynlpir.nlpir` for more information.

The sections below do not provide a comprehensive explanation of how to use NLPIR. NLPIR has its own documentation. The section below provides basic information about how to get started with PyNLPIR assuming you are familiar with NLPIR. If you're not, be sure to check out the documentation linked to below.

### Initializing and Exiting the API

Before you can call any other NLPIR functions, you need to initialize the NLPIR API. This is done by calling `Init()`. You have to specify where NLPIR's `Data` directory is. PyNLPIR ships with a copy and it's found in the top-level of the package directory. So, you can use the module-level constant `PACKAGE_DIR` when calling `Init()`:

```
from pynlpir import nlpir

nlpir.Init(nlpir.PACKAGE_DIR)
```

NLPIR defaults to using GBK encoding. If you don't plan on passing around GBK-encoded strings, you'll want to change the encoding when calling `Init()`:

```
nlpir.Init(nlpir.PACKAGE_DIR, nlpir.UTF8_CODE)
```

Once NLPIR is initialized, you can begin using the rest of the NLPIR functions. When you're finished, it's good to call `Exit()` in order to exit the NLPIR API and free the allocated memory:

```
nlpir.Exit()
```

### The Rest of the NLPIR Functions

For a complete list of NLPIR functions that `pynlpir.nlpir` includes, check out the *PyNLPIR API*. NLPIR's documentation is included in a PDF file in the NLPIR/ICTCLAS 2016 download. Consult it for detailed information on how to use NLPIR.

## What's Next

Now that you've finished the tutorial, you should be able to perform basic tasks using PyNLPIR. If you need more information regarding a module, constant, or function, be sure to check out the *PyNLPIR API*. If you need help, spot a bug, or have a feature request, then please visit PyNLPIR's GitHub Issues page.

# PyNLPIR API

## pynlpir

Provides an easy-to-use Python interface to NLPIR/ICTCLAS.

The functions below are not as extensive as the full set of functions exported by NLPIR (for that, see `pynlpir.nlpir`). A few design choices have been made with these functions as well, e.g. they have been renamed and their output is formatted differently.

The functions in this module all assume input is either unicode or encoded using the encoding specified when `open()` is called. These functions return unicode strings.

After importing this module, you must call `open()` in order to initialize the NLPIR API. When you're done using the NLPIR API, call `close()` to exit the API.

`pynlpir.`**`ENCODING`**
: The encoding configured by `open()`.

`pynlpir.`**`ENCODING_ERRORS`**
: The encoding error handling scheme configured by `open()`.

**class** pynlpir.**LicenseError**
Raised when the license is missing or expired.

pynlpir.**open**(*data_dir=nlpir.PACKAGE_DIR*, *encoding=ENCODING*, *encoding_errors=ENCODING_ERRORS*, *license_code=None*)
Initializes the NLPIR API.

This calls the function `Init()`.

**Parameters**

- **data_dir** (*str*) – The absolute path to the directory that has NLPIR's *Data* directory (defaults to `pynlpir.nlpir.PACKAGE_DIR`).
- **encoding** (*str*) – The encoding that the Chinese source text will be in (defaults to `'utf_8'`). Possible values include `'gbk'`, `'utf_8'`, or `'big5'`.
- **encoding_errors** (*str*) – The desired encoding error handling scheme. Possible values include `'strict'`, `'ignore'`, and `'replace'`. The default error handler is 'strict' meaning that encoding errors raise `ValueError` (or a more codec specific subclass, such as `UnicodeEncodeError`).
- **license_code** (*str*) – The license code that should be used when initializing NLPIR. This is generally only used by commercial users.

**Raises**

- **RuntimeError** – The NLPIR API failed to initialize. Sometimes, NLPIR leaves an error log in the current working directory or NLPIR's `Data` directory that provides more detailed messages (but this isn't always the case).
- **LicenseError** – The NLPIR license appears to be invalid or expired.

pynlpir.**close**()
Exits the NLPIR API and frees allocated memory. This calls the function `Exit()`.

pynlpir.**segment**(*s*, *pos_tagging=True*, *pos_names='parent'*, *pos_english=True*)
Segment Chinese text *s* using NLPIR.

The segmented tokens are returned as a list. Each item of the list is a string if *pos_tagging* is *False*, e.g. `['', '', ...]`. If *pos_tagging* is *True*, then each item is a tuple (`(token, pos)`), e.g. `[('', 'pronoun'), ('', 'verb'), ...]`.

If *pos_tagging* is *True* and a segmented word is not recognized by NLPIR's part of speech tagger, then the part of speech code/name will be returned as `None` (e.g. a space returns as `(' ', None)`).

This uses the function `ParagraphProcess()` to segment *s*.

**Parameters**

- **s** – The Chinese text to segment. *s* should be Unicode or a UTF-8 encoded string.
- **pos_tagging** (*bool*) – Whether or not to include part of speech tagging (defaults to `True`).
- **pos_names** (`str` or `None`) – What type of part of speech names to return. This argument is only used if *pos_tagging* is `True`. `None` means only the original NLPIR part of speech code will be returned. Other than `None`, *pos_names* may be one of `'parent'`, `'child'`, or `'all'`. Defaults to `'parent'`. `'parent'` indicates that only the most generic name should be used, e.g. `'noun'` for `'nsf'`. `'child'` indicates that the most specific name should be used, e.g. `'transcribed toponym'` for `'nsf'`. `'all'` indicates that all names should be used, e.g. `'noun:toponym:transcribed toponym'` for `'nsf'`.
- **pos_english** (*bool*) – Whether to use English or Chinese for the part of speech names, e.g. `'conjunction'` or `''`. Defaults to `True`. This is only used if *pos_names* is `True`.

pynlpir.**get_key_words**(*s*, *max_words=50*, *weighted=False*)
Determines key words in Chinese text *s*.

The key words are returned in a list. If *weighted* is `True`, then each list item is a tuple: `(word, weight)`, where *weight* is a float. If it's *False*, then each list item is a string.

This uses the function `GetKeyWords()` to determine the key words in *s*.

**Parameters**

- **s** – The Chinese text to analyze. *s* should be Unicode or a UTF-8 encoded string.
- **max_words** (*int*) – The maximum number of key words to find (defaults to `50`).
- **weighted** (*bool*) – Whether or not to return the key words' weights (defaults to `True`).

## pynlpir.nlpir

This module uses `ctypes` to provide a Python API to NLPIR. Other than argument names used in this documentation, the functions are left the same as they are in NLPIR.

When this module is imported, the NLPIR library is imported and the functions listed below are exported by a `ctypes.CDLL` instance.

There is a less extensive, easier-to-use NLPIR interface directly in the `pynlpir` module.

`Init()` must be called before any other NLPIR functions can be called. After using the API, you can call `Exit()` to exit the API and free up allocated memory.

pynlpir.nlpir.**PACKAGE_DIR**
The absolute path to this package (used by NLPIR to find its `Data` directory). This is a string in Python 2 and a bytes object in Python 3 (so it can be used with the `Init()` function below).

pynlpir.nlpir.**LIB_DIR**
The absolute path to this path's lib directory.

pynlpir.nlpir.**libNLPIR**
A `ctypes.CDLL` instance for the NLPIR API library.

pynlpir.nlpir.**GBK_CODE** 0
NLPIR's GBK encoding constant.

pynlpir.nlpir.**UTF8_CODE** 1
NLPIR's UTF-8 encoding constant.

pynlpir.nlpir.**BIG5_CODE** 2
NLPIR's BIG5 encoding constant.

pynlpir.nlpir.**GBK_FANTI_CODE** 3
NLPIR's GBK (Traditional Chinese) encoding constant.

pynlpir.nlpir.**ICT_POS_MAP_SECOND** 0
ICTCLAS part of speech constant #2.

pynlpir.nlpir.**ICT_POS_MAP_FIRST** 1
ICTCLAS part of speech constant #1.

pynlpir.nlpir.**PKU_POS_MAP_SECOND** 2
PKU part of speech constant #2.

pynlpir.nlpir.**PKU_POS_MAP_FIRST** 3
PKU part of speech constant #1.

**class** pynlpir.nlpir.**ResultT**
The NLPIR result_t structure. Inherits from ctypes.Structure.

**start**
The start position of the word in the source Chinese text string.

**length**
The detected word's length.

**sPOS**
A string representing the word's part of speech.

**word_type**
If the word is found in the user's dictionary.

**weight**
The weight of the detected word.

pynlpir.nlpir.**get_func**(*name*, *argtypes=None*, *restype=c_int*, *lib=libNLPIR*)
Retrieves the corresponding NLPIR function.

**Parameters**

- **name** (*str*) – The name of the NLPIR function to get.
- **argtypes** (*list*) – A list of ctypes data types that correspond to the function's argument types.
- **restype** – A ctypes data type that corresponds to the function's return type (only needed if the return type isn't ctypes.c_int).
- **lib** – A ctypes.CDLL instance for the NLPIR API library where the function will be retrieved from (defaults to libNLPIR).

**Returns** The exported function. It can be called like any other Python callable.

pynlpir.nlpir.**Init**(*data_dir*, *encoding=GBK_CODE*, *license_code=None*)
Initializes the NLPIR API. This must be called before any other NLPIR functions will work.

**Parameters**

- **data_dir** (*str*) – The path to the NLPIR data folder's parent folder. PACKAGE_DIR can be used for this.
- **encoding** (*int*) – Which encoding NLPIR should expect. GBK_CODE, UTF8_CODE, BIG5_CODE, and GBK_FANTI_CODE should be used for this argument.
- **license_code** (*str*) – A license code for unlimited usage. Most users shouldn't need to use this.

**Returns** Whether or not the function executed successfully.

**Return type** bool

pynlpir.nlpir.**Exit**()
Exits the NLPIR API and frees allocated memory.

**Returns** Whether or not the function executed successfully.

**Return type** bool

pynlpir.nlpir.**ParagraphProcess**(*s*, *pos_tagging=True*)
Segments a string of Chinese text (encoded using the encoding specified when Init() was called).

**Parameters**

- **s** (*str*) – The Chinese text to process.
- **pos_tagging** (*bool*) – Whether or not to return part of speech tags with the segmented words..

**Returns** The segmented words.

**Return type** str

pynlpir.nlpir.**ParagraphProcessA**(*s*, *size_pointer*, *user_dict=True*)

Segments a string of Chinese text (encoded using the encoding specified when `Init()` was called).

Here is an example of how to use this function:

```
size = ctypes.c_int()
result = ParagraphProcessA(s, ctypes.byref(size), False)
result_t_vector = ctypes.cast(result, ctypes.POINTER(ResultT))
words = []
for i in range(0, size.value):
    r = result_t_vector[i]
    word = s[r.start:r.start+r.length]
    words.append((word, r.sPOS))
```

**Parameters**

- **s** (*str*) – The Chinese text to process.
- **size_pointer** – A pointer to a `ctypes.c_int` that will be set to the result vector's size.
- **user_dict** (*bool*) – Whether or not to use the user dictionary.

**Returns** A pointer to the result vector. Each result in the result vector is an instance of `ResultT`.

pynlpir.nlpir.**FileProcess**(*source_filename*, *result_filename*, *pos_tagging=True*)

Processes a text file.

**Parameters**

- **source_filename** (*str*) – The name of the file that contains the source text.
- **result_filename** (*str*) – The name of the file where the results should be written.
- **pos_tagging** (*bool*) – Whether or not to include part of speech tags in the output.

**Returns** If the function executed successfully, the processing speed is returned (`float`). Otherwise, `0` is returned.

pynlpir.nlpir.**ImportUserDict**(*filename*)

Imports a user-defined dictionary from a text file.

**Parameters** **filename** (*str*) – The filename of the user's dictionary file.

**Returns** The number of lexical entries successfully imported.

**Return type** int

pynlpir.nlpir.**AddUserWord**(*word*)

Adds a word to the user's dictionary.

**Parameters** **word** (*str*) – The word to add to the dictionary.

**Returns** `1` if the word was added successfully, otherwise `0`.

**Return type** int

pynlpir.nlpir.**SaveTheUsrDic**()
Writes the user's dictionary to disk.

**Returns** 1 if the dictionary was saved successfully, otherwise 0.

**Return type** int

pynlpir.nlpir.**DelUsrWord**(*word*)
Deletes a word from the user's dictionary.

**Parameters** **word** (*str*) – The word to delete.

**Returns** -1 if the word doesn't exist in the dictionary. Otherwise, the pointer to the word deleted.

**Return type** int

pynlpir.nlpir.**GetKeyWords**(*s*, *max_words=50*, *weighted=False*)
Extracts key words from a string of Chinese text.

**Parameters**

- **s** (*str*) – The Chinese text to process.
- **max_words** (*int*) – The maximum number of key words to return.
- **weighted** (*bool*) – Whether or not the key words' weights are returned.

**Returns** The key words.

**Return type** str

pynlpir.nlpir.**GetFileKeyWords**(*filename*, *max_words=50*, *weighted=False*)
Extracts key words from Chinese text in a file.

**Parameters**

- **filename** (*str*) – The file to process.
- **max_words** (*int*) – The maximum number of key words to return.
- **weighted** (*bool*) – Whether or not the key words' weights are returned.

**Returns** The key words.

**Return type** str

pynlpir.nlpir.**GetNewWords**(*s*, *max_words=50*, *weighted=False*)
Extracts new words from a string of Chinese text.

**Parameters**

- **s** (*str*) – The Chinese text to process.
- **max_words** (*int*) – The maximum number of new words to return.
- **weighted** (*bool*) – Whether or not the new words' weights are returned.

**Returns** The new words.

**Return type** str

pynlpir.nlpir.**GetFileNewWords**(*filename*, *max_words=50*, *weighted=False*)
Extracts new words from Chinese text in a file.

**Parameters**

- **filename** (*str*) – The file to process.
- **max_words** (*int*) – The maximum number of new words to return.

- **weighted** (*bool*) – Whether or not the new words' weights are returned.

**Returns** The new words.

**Return type** str

pynlpir.nlpir.**FingerPrint**(*s*)
Extracts a fingerprint from a string of Chinese text.

**Parameters** **s** (*str*) – The Chinese text to process.

**Returns** The fingerprint of the content. `0` if the function failed.

pynlpir.nlpir.**SetPOSmap**(*pos_map*)
Selects which part of speech map to use.

**Parameters** **pos_map** (*int*) – The part of speech map that should be used. This should be one of `ICT_POS_MAP_FIRST`, `ICT_POS_MAP_SECOND`, `PKU_POS_MAP_FIRST`, or `PKU_POS_MAP_SECOND`.

**Returns** `0` if the function failed, otherwise `1`.

**Return type** int

pynlpir.nlpir.**NWI_Start**()
Initializes new word identification.

**Returns** `True` if the function succeeded; `False` if it failed.

**Return type** bool

pynlpir.nlpir.**NWI_AddFile**(*filename*)
Adds the words in a text file.

**Parameters** **filename** (*string*) – The text file's filename.

**Returns** `True` if the function succeeded; `False` if it failed.

**Return type** bool

pynlpir.nlpir.**NWI_AddMem**(*filename*)
Increases the allotted memory for new word identification.

**Parameters** **filename** (*string*) – NLPIR's documentation is unclear on what this argument is for.

**Returns** `True` if the function succeeded; `False` if it failed.

**Return type** bool

pynlpir.nlpir.**NWI_Complete**()
Terminates new word identifcation. Frees up memory and resources.

**Returns** `True` if the function succeeded; `False` if it failed.

**Return type** bool

pynlpir.nlpir.**NWI_GetResult**(*weight*)
Returns the new word identification results.

**Parameters** **weight** (*bool*) – Whether or not to include word weights in the results.

**Returns** `True` if the function succeeded; `False` if it failed.

**Returns** The identified words.

**Return type** str

pynlpir.nlpir.**NWI_Results2UserDict**()
Adds the newly identified words to the user dictionary.

This function should only be called after `NWI_Complete()` is called.

If you want to save the user dictionary, consider running `SaveTheUsrDic()`.

**Returns** `1` if the function succeeded; `0` if it failed.

**Return type** int

### pynlpir.pos_map

Part of speech mapping constants and functions for NLPIR/ICTCLAS.

This module is used by `pynlpir` to format segmented words for output.

pynlpir.pos_map.**POS_MAP**
A dictionary that maps part of speech codes returned by NLPIR to human-readable names (English and Chinese).

pynlpir.pos_map.**get_pos_name**(*code*, *name='parent'*, *english=True*)
Gets the part of speech name for *code*.

**Parameters**

- **code** (*str*) – The part of speech code to lookup, e.g. `'nsf'`.
- **name** (*str*) – Which part of speech name to include in the output. Must be one of `'parent'`, `'child'`, or `'all'`. Defaults to `'parent'`. `'parent'` indicates that only the most generic name should be used, e.g. `'noun'` for `'nsf'`. `'child'` indicates that the most specific name should be used, e.g. `'transcribed toponym'` for `'nsf'`. `'all'` indicates that all names should be used, e.g. `('noun', 'toponym', 'transcribed toponym')` for `'nsf'`.
- **english** (*bool*) – Whether to return an English or Chinese name.

**Returns** `str` (`unicode` for Python 2) if *name* is `'parent'` or `'child'`. `tuple` if *name* is `'all'`. `None` if the part of speech code is not recognized.

# 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:

## Types of Contributions

### Report Bugs

Report bugs at https://github.com/tsroten/pynlpir/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

PyNLPIR could always use more documentation, whether as part of the official PyNLPIR 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/pynlpir/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 :)

## Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/pynlpir.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 pynlpir
   $ cd pynlpir/
   $ 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 pynlpir
   $ 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.

### 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 3.4. Check https://travis-ci.org/tsroten/pynlpir/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*.

## Change Log

### 0.5.2 (2017-03-24)

- Fixes error opening PyNLPIR on Windows/Python 3. Fixes #63.

### 0.5.1 (2017-03-18)

- Fixes error opening PyNLPIR on Windows/Python 2. Fixes #63.

### 0.5 (2017-03-11)

- Adds license auto-updater.
- Removes included license file.
- Moves tests to separate directory.

### 0.4.6 (2016-08-12)

- Updates NLPIR license.

### 0.4.5 (2016-07-18)

- Updates NLPIR license.

### 0.4.4 (2016-04-09)

- Updates NLPIR license.

### 0.4.3 (2016-03-13)

- Updates NLPIR license.

### 0.4.2 (2016-02-16)

- Updates NLPIR license.

### 0.4.1 (2016-01-22)

- Updates NLPIR license.

### 0.4 (2015-12-21)

- Updates NLPIR.
- Adds OS X support.

### 0.3.3 (2015-10-21)

- Fixes NLPIR freezing with certain inputs. Fixes #33.
- Adds flake8 tests to tox and travis-ci.
- Adds tests for Python 3.5 support.
- Uses io.open() in setup.py. Fixes #34.

### 0.3.2 (2015-08-05)

- Adds 2015-08-05 license file. Fixes #31.

### 0.3.1 (2015-07-12)

- Fixes RST rendering error.

### 0.3 (2015-07-12)

- Includes NLPIR version 20150702. Fixes #25.
- Adds encoding error handling scheme options.
- Adds new word identification functions and documentation. Fixes #26.
- Makes `~pynlpir.get_key_words` work with multiple NLPIR return value structures. Fixes #23.
- Returns `None` when pos code not recognized. Fixes #20.
- Updates outdated link in tutorial. Fixes #21.

### 0.2.2 (2015-01-02)

- Fixes release problem with v0.2.1.

### 0.2.1 (2015-01-02)

- Packages NLPIR version 20141230. Fixes #18.

### 0.2 (2014-12-18)

- Packages NLPIR version 20140926. Restores `pynlpir.get_key_words` functionality. Fixes #11, #14, and #15.
- Updates part of speech map for new NLPIR version. Fixes #17.
- Fixes a typo in `api.rst`. Fixes #16.
- Fixes a bug involving uppercase part of speech codes. Fixes #10.
- Adds Python 3.4 test to tox and travis.
- Notes Python 3.4 support in `setup.py` and `CONTRIBUTING.rst`.
- Fixes the doubleslash unit test so that it works with the new NLPIR version.
- Adds a missing comma. Fixes #8.
- Fixes indentation in `pynlpir.get_key_words`.
- Adds condition for empty key word result. Fixes #9.

### 0.1.3 (2014-06-12)

- Fixes typo in docs. Fixes #4.
- Adds *license_code* argument to `pynlpir.open`. Fixes #6.
- Packages NLPIR version 20131219 and removes version 20140324. Fixes a NLPIR expired license issue. Fixes #5.
- Fixes bug with double slashes in input. Fixes #7.

### 0.1.2 (2014-05-01)

- Adds version information to `__init__.py`.
- Adds Travis CI configuration information.
- Reformats `README.rst`.
- Adds documentation about contributing.
- Fixes #2. Fixes segmenting text with whitespace.
- Fixes #3. Fixes `_encode()`/`_decode` default encoding error.

### 0.1.1 (2014-04-07)

- Fixes installation problem with package data.

### 0.1.0 (2014-04-07)

- Initial release.

## Credits

### Author and Maintainer

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

### Contributors

None yet. Why not be the first?

## p