Zhon Documentation

Release 1.1.5

Thomas Roten

Mar 12, 2017

Contents

CHAPTER 1

Introduction

Zhon is a Python library that provides constants commonly used in Chinese text processing:

- CJK characters and radicals
- Chinese punctuation marks
- Chinese sentence regular expression pattern
- Pinyin vowels, consonants, lowercase, uppercase, and punctuation
- Pinyin syllable, word, and sentence regular expression patterns
- Zhuyin characters and marks
- Zhuyin syllable regular expression pattern
- CC-CEDICT characters

CHAPTER 2

Installation

Zhon supports Python 2.7 and 3. Install using pip:

```
$ pip install zhon
```

If you want to download the latest source code, check out Zhon's GitHub repository.

Be sure to report any bugs you find. Thanks!

CHAPTER 3

Using Zhon

Zhon contains four modules that export helpful Chinese constants:

- *zhon.hanzi*
- *zhon.pinyin*
- *zhon.zhuyin*
- *zhon.cedict*

Zhon's constants are formatted in one of three ways:

- Characters listed individually. These can be used with membership tests or used to build regular expression patterns. For example, `'aeiou'`.
- Character code ranges. These are used to build regular expression patterns. For example, `'u\0041-\u005A\u0061-\u007A'`.
- Regular expression pattern. These are regular expression patterns that can be used with the regular expression library directly. For example, `'[u\0020-\u007E]+'`.

Using the constants listed below is simple. For constants that list the characters individually, you can perform membership tests or use them in regular expressions:

```
>>> '' in zhon.cedict.traditional
False

>>> # This regular expression finds all characters that aren't considered
... # traditional according to CC-CEDICT
... re.findall('[^{}]'.format(zhon.cedict.traditional), '')
['', '', '']
```

For constants that contain character code ranges, you'll want to build a regular expression:

```
>>> re.findall('[{}]'.format(zhon.hanzi.punctuation), '')
['']
```

For constants that are regular expression patterns, you can use them directly with the regular expression library, without formatting them:

```
>>> re.findall(zhon.hanzi.sentence, '')
['', '']
```

zhon.hanzi

These constants can be used when working directly with Chinese characters.

These constants can be used in a variety of ways, but they can't directly distinguish between Chinese, Japanese, and Korean characters/words. Chapter 12 of The Unicode Standard (PDF) has some useful information about this:

> There is some concern that unifying the Han characters may lead to confusion because they are sometimes used differently by the various East Asian languages. Computationally, Han character unification presents no more difficulty than employing a single Latin character set that is used to write languages as different as English and French. Programmers do not expect the characters "c", "h", "a", and "t" alone to tell us whether chat is a French word for cat or an English word meaning "informal talk." Likewise, we depend on context to identify the American hood (of a car) with the British bonnet. Few computer users are confused by the fact that ASCII can also be used to represent such words as the Welsh word ynghyd, which are strange looking to English eyes. Although it would be convenient to identify words by language for programs such as spell-checkers, it is neither practical nor productive to encode a separate Latin character set for every language that uses it.

`zhon.hanzi.`**`characters`**
`zhon.hanzi.`**`cjk_ideographs`**

Character codes and code ranges for pertinent CJK ideograph Unicode characters. This includes:

- CJK Unified Ideographs
- CJK Unified Ideographs Extension A
- CJK Unified Ideographs Extension B
- CJK Unified Ideographs Extension C
- CJK Unified Ideographs Extension D
- CJK Compatibility Ideographs
- CJK Compatibility Ideographs Supplement
- Ideographic number zero

Some of the characters in this constant will not be Chinese characters, but this is a convienient way to approach the issue. If you'd rather have an enormous string of Chinese characters from a Chinese dictionary, check out *`zhon.cedict`*.

`zhon.hanzi.`**`radicals`**

Character code ranges for the Kangxi Radicals and CJK Radicals Supplement Unicode blocks.

`zhon.hanzi.`**`punctuation`**

This is the concatenation of *`zhon.hanzi.non_stops`* and *`zhon.hanzi.stops`*.

`zhon.hanzi.`**`non_stops`**

The string `'–––‵′“”„....'`. This contains Chinese punctuation marks, excluding punctuation marks that function as stops.

`zhon.hanzi.`**`stops`**

The string `''`. These punctuation marks function as stops.

`zhon.hanzi.`**`sent`**
`zhon.hanzi.`**`sentence`**

A regular expression pattern for a Chinese sentence. A sentence is defined as a series of CJK characters (as defined by *`zhon.hanzi.characters`*) and non-stop punctuation marks followed by a stop and zero or more container-closing punctuation marks (e.g. apostrophe and brackets).

```
>>> re.findall(zhon.hanzi.sentence, '')
['']
```

`zhon.pinyin`

These constants can be used when working with Pinyin.

`zhon.pinyin.`**`vowels`**

The string `'aeiouvüāēīōūǖáéíóúǘǎěǐǒǔǚàèìòùǜAEIOUVÜĀĒĪŌŪǕÁÉÍÓÚǗǍĚǏǑǓǙÀÈÌÒÙǛ'`. This contains every Pinyin vowel (lowercase and uppercase).

`zhon.pinyin.`**`consonants`**

The string `'bpmfdtnlgkhjqxzcsrwyBPMFDTNLGKHJQXZCSRWY'`. This contains every Pinyin consonant (lowercase and uppercase).

`zhon.pinyin.`**`lowercase`**

The string `'bpmfdtnlgkhjqxzcsrwyaeiouvüāēīōūǖáéíóúǘǎěǐǒǔǚàèìòùǜ'`. This contains every lowercase Pinyin vowel and consonant.

`zhon.pinyin.`**`uppercase`**

The string `'BPMFDTNLGKHJQXZCSRWYAEIOUVÜĀĒĪŌŪǕÁÉÍÓÚǗǍĚǏǑǓǙÀÈÌÒÙǛ'`. This contains every uppercase vowel and consonant.

`zhon.pinyin.`**`marks`**

The string `"·012345:-'"`. This contains all Pinyin marks that have special meaning: a middle dot and numbers for indicating tone, a colon for easily writing ü ('u:'), a hyphen for connecting syllables within words, and an apostrophe for separating a syllable beginning with a vowel from the previous syllable in its word. All of these marks can be used within a valid Pinyin word.

`zhon.pinyin.`**`punctuation`**

The concatenation of *`zhon.pinyin.non_stops`* and *`zhon.pinyin.stops`*.

`zhon.pinyin.`**`non_stops`**

The string `'"#$%&\'()*+,-/:;<=>@[\]^_`{|}~"'`. This contains every ASCII punctuation mark that doesn't function as a stop.

`zhon.pinyin.`**`stops`**

The string `'.!?'`. This contains every ASCII punctuation mark that functions as a stop.

`zhon.pinyin.`**`printable`**

The concatenation of *`zhon.pinyin.vowels`*, *`zhon.pinyin.consonants`*, *`zhon.pinyin.marks`*, *`zhon.pinyin.punctuation`*, and `string.whitespace`. This is essentially a Pinyin whitelist for complete Pinyin sentences – it's every possible valid character a Pinyin string can use assuming all non-Chinese words that might be included (like proper nouns) use ASCII.

Validating and splitting Pinyin isn't as simple as checking that only valid characters exist or matching maximum-length valid syllables. The regular expression library's lookahead features are used in this module's regular expression patterns to ensure that only valid Pinyin syllables are matched. The approach used to segment a string into valid Pinyin syllables is roughly:

1. Match the longest possible valid syllable.

2. If that match is followed directly by a vowel, drop that match and try again with the next longest possible valid syllable.

Additionally, lookahead assertions are used to ensure that hyphens and apostrophes are only accepted when they are used correctly. This helps to weed out non-Pinyin strings.

zhon.pinyin.**syl**
zhon.pinyin.**syllable**

A regular expression pattern for a valid Pinyin syllable (accented or numbered). Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.syllable, 'Shū zài zhuōzi shàngmian. Shu1 zai4␣
↪zhuo1zi5 shang4mian5.', re.IGNORECASE)
['Shū', 'zài', 'zhuō', 'zi', 'shàng', 'mian', 'Shu1', 'zai4', 'zhuo1', 'zi5',
↪'shang4', 'mian5']
```

zhon.pinyin.**a_syl**
zhon.pinyin.**acc_syl**
zhon.pinyin.**accented_syllable**

A regular expression for a valid accented Pinyin syllable. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.acc_syl, 'Shū zài zhuōzi shàngmian.', re.IGNORECASE)
['Shū', 'zài', 'zhuō', 'zi', 'shàng', 'mian']
```

zhon.pinyin.**n_syl**
zhon.pinyin.**num_syl**
zhon.pinyin.**numbered_syllable**

A regular expression for a valid numbered Pinyin syllable. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.num_syl, 'Shu1 zai4 zhuo1zi5 shang4mian5.', re.
↪IGNORECASE)
['Shu1', 'zai4', 'zhuo1', 'zi5', 'shang4', 'mian5']
```

zhon.pinyin.**word**

A regular expression pattern for a valid Pinyin word (accented or numbered). Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.word, 'Shū zài zhuōzi shàngmian. Shu1 zai4 zhuo1zi5␣
↪shang4mian5.', re.IGNORECASE)
['Shū', 'zài', 'zhuōzi', 'shàngmian', 'Shu1', 'zai4', 'zhuo1zi5', 'shang4mian5'
```

zhon.pinyin.**a_word**
zhon.pinyin.**acc_word**
zhon.pinyin.**accented_word**

A regular expression for a valid accented Pinyin word. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.acc_word, 'Shū zài zhuōzi shàngmian.', re.IGNORECASE)
['Shū', 'zài', 'zhuōzi', 'shàngmian']
```

zhon.pinyin.**n_word**
zhon.pinyin.**num_word**
zhon.pinyin.**numbered_word**

A regular expression for a valid numbered Pinyin word. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.num_word, 'Shu1 zai4 zhuo1zi5 shang4mian5.', re.
↪IGNORECASE)
['Shu1', 'zai4', 'zhuo1zi5', 'shang4mian5']
```

zhon.pinyin.**sent**
zhon.pinyin.**sentence**

A regular expression pattern for a valid Pinyin sentence (accented or numbered). Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.sentence, 'Shū zài zhuōzi shàngmian. Shu1 zai4␣
↪zhuo1zi5 shang4mian5.', re.IGNORECASE)
['Shū zài zhuōzi shàngmian.', 'Shu1 zai4 zhuo1zi5 shang4mian5.']
```

zhon.pinyin.**a_sent**
zhon.pinyin.**acc_sent**
zhon.pinyin.**accented_sentence**

A regular expression for a valid accented Pinyin sentence. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.acc_sent, 'Shū zài zhuōzi shàngmian.', re.IGNORECASE)
['Shū zài zhuōzi shàngmian.']
```

zhon.pinyin.**n_sent**
zhon.pinyin.**num_sent**
zhon.pinyin.**numbered_sentence**

A regular expression for a valid numbered Pinyin sentence. Compile with `re.IGNORECASE` (`re.I`) to accept uppercase letters as well.

```
>>> re.findall(zhon.pinyin.num_sent, 'Shu1 zai4 zhuo1zi5 shang4mian5.', re.
↪IGNORECASE)
['Shu1 zai4 zhuo1zi5 shang4mian5.']
```

zhon.zhuyin

These constants can be used when working with Zhuyin (Bopomofo).

zhon.zhuyin.**characters**

The string `''`. This contains all Zhuyin characters as defined by the Bomopofo Unicode block. It does not include the Bomopofo Extended block that defines characters used in non-standard dialects or minority languages.

zhon.zhuyin.**marks**

The string `'ˇ'`. This contains the Zhuyin tone marks.

zhon.zhuyin.**syl**
zhon.zhuyin.**syllable**

A regular expression pattern for a valid Zhuyin syllable.

```
>>> re.findall(zhon.zhuyin.syllable, '    ')
['', '', '', '']
```

`zhon.cedict`

These constants are built from the CC-CEDICT dictionary. They aren't guaranteed to contain every possible Chinese character. They only provide characters that exist in the CC-CEDICT dictionary.

`zhon.cedict.`**`all`**

A string containing all Chinese characters found in CC-CEDICT.

`zhon.cedict.`**`trad`**

`zhon.cedict.`**`traditional`**

A string containing characters considered by CC-CEDICT to be Traditional Chinese characters. Some of these characters are also present in *`zhon.cedict.simplified`* because many characters were left untouched by the simplification process.

`zhon.cedict.`**`simp`**

`zhon.cedict.`**`simplified`**

A string containing characters considered by CC-CEDICT to be Simplified Chinese characters. Some of these characters are also present in *`zhon.cedict.traditional`* because many characters were left untouched by the simplification process.

Python Module Index

Index