kconv/kre/kstring モジュールの使用の手引き

Abstract

唐沢さんによる Python での日本語処理支援モジュール kconv/kre/kstring の使用法を解説。

`http://www.tomigaya.shibuya.tokyo.jp/~mak/`

を参照のこと。

1 始めに

kconv/kstring/kre は唐沢氏による日本語処理支援のための Python モジュールです。kconv には C 版と純粋に Python のみでかかれた版の二種類があります。

kconv ECU/JIS/SJIS 等の日本語コード間の相互変換をおこないます。

kstring 日本語文字列を表す kstring オブジェクトを提供します。kstring オブジェクトは Python の string オブジェクトと同じように、スライシングなどの操作を日本語文字列に対して提供します。

kre 日本語対応の正規表現処理モジュールです。

2 kconv

2.1 kconv について

kconv は唐沢氏による Python での日本語処理支援のためのモジュールの一つです。kconv はその中で、EUC,JIS, Shift-JIS 等の日本語コード相互間の変換を行うためのモジュールです。日本語エンコーディング間の変更と同時に行末文字 (CR, CR+LF, LF) の変更も行います。また半角仮名文字を含む文字列を処理し、全角仮名文字に変更することができます。C++版は Python のみで書かれた版より高速な処理が可能です。C++版/Python 版のどちらでも同じ API を使用することができます。

2.2 モジュールのインストール

C++版では次の手順で kconv モジュールをインストールします。

```
> tar xzvf kconv-1.1.7c.tar.gz
        > cd kconv-1.1.7c
        > ./configure [--enable-optimize]
        > make
        > su
        # make install
```

もし、Python の最適化を望むのなら、

```
    >./configure --enable-optimize
```

にかわって、

```
    > ./configure
```

を使う必要があります。
　kconv モジュールができ上がれば、これを PYTHONPATH 中にあるいずれかのディレクトリに移動することで、kconv モジュールが利用可能となります。

```
ex.)
    > tar xzvf kconv-0.5.5p.tar.gz
    > cd kconv-0.5.5p
    > python compile.py [optimize]
    > tar cvf - kconv |
            (cd /usr/local/lib/python1.5/site-packages/;tar xvpf -)
    NOTE: preceding command must be one line
```

2.3　kconv モジュール内の関数

kconv モジュールの利用に先立っては、

```
>> import kconv
```

を実行しておきます。kconv モジュールでは、kconv オブジェクトを作成するための関数 Kconv が定義されています。Kconv 関数の呼び出し形式は Python の関数定義に準じると、

```
kconv.Kconv(outcode = DEFAULT_OUTPUT_CODING,incode = AUTO
,hankanaconvert = ZENKAKU ,checkmode = TABLE
                                ,mode =WHOLE , blcode=DEFAULT_BREAKLINE_CODE)
```

とかけます。全ての引数にデフォルト値が書かれているので、引数なしの呼び出し

```
kc=kconv.Kconv()
```

も意味を持っています。

out-code 出力のエンコーディングを指定します。省略時の既定値は defaults.py で定義されている DEFAULT_OUTPUT_CODING です。

kconv.EUC - EUC

kconv.SJIS - Shift Jis

kconv.JIS - Jis

kconv.UNICODE - Unicode(UCS2)

kconv.UTF8 - UTF-8

in-code 入力データのエンコーディングが判っている場合には、より良い実行効率のために入力コードを指定しましょう。入力のコード体系が指定されていない場合には、自動認識の機能が働きます。

kconv.EUC - EUC

kconv.SJIS - Shift Jis

kconv.JIS - Jis

kconv.UNICODE - Unicode(UCS2)

kconv.UTF8 - UTF-8

kconv.AUTO - 自動認識 (省略時の既定値)

hankanaconvert kconv が半角カタカナを全角かたかなに変換するかどうかを指定します。 出力コード体系が EUC でかつ半角カタカナを全角カタカナに変換することを望まないときには、kconv は EUC における 2 バイトの半角カタカナを出力します。

kconv.ZENKAKU - カタカナを全角かたかなに変換する (省略時の既定値)

kconv.HANKAKU - 入力に半角カタカナがあれば、半角カタカナを出力する。

mode 全ての入力文字列を一括で変換するか、一行毎に変換するかを指定します。

kconv.LINE - 一行毎に変換する。入力コード体系が一行毎に替わっている可能性のある場合や、バッファに大きなメモリを割当てたくない場合に使用する。

kconv.WHOLE - 入力文字列を一括で変換する.(省略時の既定値)

blcode 出力の行末文字を指定します。省略時の既定値は defaults.py で定義されている DEFAULT_BREAKLINE_CODE です。

kconv.LF - LF (0x0A) Unix

kconv.CR - CR (0x0D) Macintosh

kconv.CL - CR + LF (0x0D + 0x0A) Windows / DOS

2.4 使用例

最初はもっとも単純な使用例です。

```
>>> import kconv #kconv をインポートする。
>>> toEuc = kconv.Kconv()  # kconv の実体 (インスタンス) を作る
>>> print toEuc.convert( input_string )  # convert() メソッドでコード変換を行う。
```

convert() メソッドはコード変換後の文字列を値として返します。
次の例では、入力コード体系 (JIS) と出力コード体系 (SJIS) を指定しています。

```
>>> kc = kconv.Kconv(kconv.SJIS,kconv.JIS,kconv.HANKAKU)
>>> print kc.convert("JIS コードの文字列だもん")
```

最後の例では、入力コードの自動認識機能を使い EUC への変換をおこないます。合わせて、半角カタカナを全角カタカナに変換します。自動認識のアルゴリズムとして全文検索アルゴリズムを使います。

```
>>> kc = kconv.Kconv(kconv.SJIS,kconv.AUTO,kconv.ZENKAKU,
                     kconv.FULL)
>>> print kc.convert("文字コードは任意だよ。")
```

2.5 その他の機能

kconv モジュールにはコード変換以外の機能を提供する関数も用意されています。これらの関数は全て第二引数として, Kconv 関数と同じように、入力文字列のコード体系を指定できます。これを省略したときには、自動検出機能によって入力文字列のコード体系を決定します。出力コードの既定値は EUC です。

- kconv.ChkHiragana(input_string[,icode])
- kconv.ChkKatakana(input_string[,icode])

これらの関数は入力文字列に平仮名 (あるいはカタカナ)、空白文字、改行以外の文字が入っているかどうかを検査します。平仮名 (あるいはカタカナ)、空白文字、改行だけからなる入力に対しては 1 、それ以外では 0 を値として返します。

例)

```
>>>print kconv.ChkHiragana("ひらがなだけなんだもん")
1
>>>print kconv.ChkKatakana("カタカナだけじゃないみたい")
0
```

- kconv.NumberConvert(input_string[,icode])

全角文字の洋数字を半角文字の数字に変換します。

```
ex)
    >>>print kconv.NumberConvert("342 2 3 6 3 234 3 3 6 8 024")
    '34223632343361024'
```

- kconv.Han2Zen(input_string, icode)
- kconv.Zen2Han(input_stirng, icode)

平仮名〔あるいはカタカナ）をカタカナ（あるいは平仮名）に変換します。英数字は変更されません。

```
例）
    >>>print kconv.Hira2Kata("東方は赤く燃えているようです")
    東方ハ赤ク燃エテイルヨウデス
    >>>print kconv.Kata2Hira("デフォルトのコード認識ルーチン")
    でふぉるとのこーど認識るーちん
```

- kconv.Upper(input_string,icode)
- kconv.Lower(input_string,icode)

英小文字（あるいは英大文字）を英大文字（あるいは英小文字）に変換します。全角、半角のどちらの英文字も変更の対象となります。その他の文字は影響を受けません。

```
ex)
    >>>print kconv.Upper("Captain ＡＭＥＲＩＣＡ")
    CAPTAIN ＡＭＥＲＩＣＡ
    >>>print kconv.Lower("Hoe Ｕｋｙｕ")
    hoe ｕｋｙｕ
```

2.6 注記

- kconv は最新の GPL に基づいて配付されています。
- kconv は EUC, JIS, SJIS, Unicode(UCS2), UTF-8 の各コード体系を取り扱います。
- kconv の使用する定数は C++版と Python 版では異なっています。
- kconv.FAST はそれほど早く無いうえに、時々は正しくコード体系を検出できないことがあります。
- 行毎変換のモードで、kconv が Unicode を検出すると、kconv はその後の全ての入力行は Unicode を使っていると仮定します。これは kconv が Unicode を検出する際に、その他のコード体系では使われていない 0x00 と 0xFF を使っているためです。その他のコード体系の入力文字はヌル文字 (0x00) を含んでいてはいけません。もし、ヌル文字が入力文字列中にあると、kconv はその文字列は Unicode の文字列であると仮定します。
- Python 版の kconv はエラーを生じた時には、kconv.KconvError 例外を発生します。

3 kstring

kstring は Python の日本語対応 string モジュールです。現在、ソースコードは kconv のソースコード (C++版) と統合されています。

3.1 なにができるの？

実行例をごらんくださいな。

```
Python 1.5.2 (#3, Aug 12 1999, 03:16:46)  [GCC 2.7.2.1] on freebsd3
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> import kstring
>>> s = kstring.kstring(open("LICENCE.EUC").readline())
>>> s
'python\xCD\xD1\xA4\xCE\xB4\xC1\xBB\xFA\xA5\xB3\xA1\xBC\xA5\xC9\xCA\xD1\xB4\xB9
\xA4\xAA\xA4\xE8\xA4\xD3\xB4\xCA\xC3\xB1\xA4\xCA\xC6\xFC\xCB\xDC\xB8\xEC\xBD\xE
8\xCD\xFD\xA5\xE9\xA5\xA4\xA5\xD6\xA5\xE9\xA5\xEA\xA1\xBC
'
>>> print s
python 用の漢字コード変換および簡単な日本語処理ライブラリー

>>> print s[10]
コ
>>> print s[10:20]
コード変換および簡単
>>> print s[-3:]
リー

>>> print s[7:13]+s[15:18]
の漢字コードおよび
>>> print s[7:13]+s[15:18]*4
の漢字コードおよびおよびおよびおよび
```

4 kre

kre は Python の日本語対応正規表現モジュールです

4.1 なにができるの？

実行例をごらんくださいな。

```
> python test.py
kre.match("あ[か-こ]*さ","あかきくけこけくきかさ").group()
あかきくけこけくきかさ

kre.search("最初（うきゅ）*最後","最初うきゅ最後ここは無視最初うきゅうきゅうきゅ最後ここも無
視").group()
最初うきゅ最後

kre.split("最初(うきゅ)*最後","最初うきゅ最後ここは無視最初うきゅうきゅうきゅ最後ここも無視")[0]
ここは無視

kre.split("最初(うきゅ)*最後","最初うきゅ最後ここは無視最初うきゅうきゅうきゅ最後ここも無視")[1]
ここも無視

kre.findall("最初（うきゅ）*最後","最初うきゅ最後ここは無視最初うきゅうきゅうきゅ最後ここも無
視")[0]
最初うきゅ最後
```

```
kre.findall("最初（うきゅ）*最後","最初うきゅ最後ここは無視最初うきゅうきゅうきゅ最後ここも無
視")[1]
最初うきゅうきゅうきゅ最後

kre.sub("い","井","いまのうちにいっぱいいいっておいたほうがいいんでないかい")
井まのうちに井っぱ井井ってお井たほうが井井んでな井か井

kre.sub("い",lambda m:"イ","いまのうちにいっぱいいいっておいたほうがいいんでないかい")
イまのうちにイっぱイイイっておイたほうがイイんでなイかイ

kre.subn("い",lambda m:"イ","いまのうちにいっぱいいいっておいたほうがいいんでないかい")
('\245\244\244\336\244\316\244\246\244\301\244\313\245\244\244\303\244\321\245\244\245
\244\244\303\244\306\244\252\245\244\244\277\244\333\244\246\244\254\245\244\245\244
\244\363\244\307\244\312\245\244\244\253\245\244', 9)
```

4.2 どうやってるの？

grep の multi byte パッチのドキュメントにあるような正規表現の翻訳を行なっています。

```
あいう -> (あ)(い)(う)
あい*う -> (あ)(い)*(う)
etc...
```

内部で 2 種類のパターンを使い分けているため kre.compile はありません。 それ以外のメソッド

```
search
match
split
findall
sub
subn
```

はオリジナルの re と同じ動きをすると思います。

A kconv man page

```
This is release 1.0- of kconv.

NAME
    kconv - Kanji code Conversion library

DESCRIPTION
    Kconv is a library for kanji code conversion in Python. A string encoded in
    any encoding can be converted into any other encoding. Any linebreak characters can
    also be converted. In addition, if hankaku katakana charaters are processed, they
    will be converted into zenkaku katakana.
    The C++ version works faster than the pure Python version.
    BUT, the pure Python version will work on any platform where Python works.
    The APIs of C++ version and those of the pure Python version are entirely identical,
```

so one can easily switch between the C++ and Python implementations with no need
rewrite source. NOTE: This file contains examples of Japanese text in the EUC encoding.
Please make sure that your viewer can display EUC.

INSTALLATION

*C++ version

```
> tar xzvf kconv-0.5.5c.tar.gz
> cd kconv-0.5.5c
> ./configure [--enable-optimize]
> make
> su
# make install
```

If you want to enable Python optimization in the compiler, use

```
>./configure --enable-optimize
```

instead of

```
> ./configure
```

Or you can use kconv without installing it as root. To do this, simply copy
the kconv files into your desired directory
(and make sure that that directory is in the PYTHONPATH????)

*Python version
After byte compiling, copy the "kconv" directory into the Python library directory
or your directory.

ex.)

```
> tar xzvf kconv-0.5.5p.tar.gz
> cd kconv-0.5.5p
> python compile.py [optimize]
> tar cvf - kconv |
        (cd /usr/local/lib/python1.5/site-packages/;tar xvpf -)
```

NOTE: preceding command must be one line

If you don't have tar, use the following command to copy the directory structure

```
> cp -R kconv /usr/local/lib/python1.5/site-packages/
```

INTRODUCTION TO THE ROUTINES

First, import "kconv" module

```
>>> import kconv
```

The following are forms of the kconv constructor
1.normal arguments

```
>>> kc = kconv.Kconv([outcode[,incode[,hankanaconvert[,checkmode
                          [,mode[,blcode]]]]]])
```

2.keyword arguments

```
>>> kc = kconv.Kconv([outcode = kconv.???][,incode = kconv.???]....)
```

Allowed Arguments
* out-code

```
Specify output encoding. The default is DEFAULT_OUTPUT_CODING in defaults.py.

    kconv.EUC       - EUC
    kconv.SJIS      - Shift Jis
    kconv.JIS       - Jis
    kconv.UNICODE   - Unicode(UCS2)
    kconv.UTF8      - UTF-8


* in-code
    If you know input encoding, specify input code for higher performance.
    If no input encoding is specified, auto-detection is automatically enabled.

    kconv.EUC       - EUC
    kconv.SJIS      - Shift Jis
    kconv.JIS       - Jis
    kconv.UNICODE   - Unicode(UCS2)
    kconv.UTF8      - UTF-8
    kconv.AUTO      - Auto detection(DEFAULT)

* hankanaconvert
    Specify whether kconv converts hankaku katakana to zenkaku katakana or
    not. If the output encoding is EUC and you do want not to convert hankaku katakna to
    zenkaku katakana, kconv outputs 2 byte hankaku katakana in EUC.

    kconv.ZENKAKU   - Convert hankaku katakana to zenkaku katakana(DEFAULT)
    kconv.HANKAKU   - Output hankaku katakana if input string includes hankaku katakana

* checkmode
    Choice auto dectection algorithms.
    If you specify input-code, checkmode is ignored.

    kconv.FAST      - Discriminate using the first definitive character.
                      (Fast and unreliable.)
    kconv.FULL      - Discriminate with whole input sting. (Weighted Average)
    kconv.TABLE     - Discriminate with a 1-byte freqency table.(DEFAULT)
    kconv.TABLE2    - Discriminate with a 2-byte freqency table.
                      The accurracy of TABLE2 is higher than TABLE, but
                      TABLE2 is slightly slower.

* mode
    Specify conversion of the whole input string or line-by-line.

    kconv.LINE      - Convert line-by-line. For cases when input code may change on a
                      line-by-line basis or when you would like to
                      use less buffer memory.
    kconv.WHOLE     - Convert whole input string.(DEFAULT)

* blcode
    Specify output breakline character
    Default breakline character is DEFAULT_BREAKLINE_CODE at defaults.py
    kconv.LF            - LF (0x0A) Unix
    kconv.CR        - CR (0x0D) Macintosh
```

```
        kconv.CL        - CR + LF (0x0D + 0x0A) Windows / DOS

EXAMPLES
    ex1) The most simple usage.
    1.Import kconv
        >>> import kconv

    2.Create instance of kconv
        >>> toEuc = kconv.Kconv()

    3.Call convert() of the instance
        >>> print toEuc.convert( input_string )

    The method convert() will return the converted string.

    ex2) Converting by specifying the input(JIS) and output(SJIS) encodings.
        >>> kc = kconv.Kconv(kconv.SJIS,kconv.JIS,kconv.HANKAKU)
        >>> print kc.convert("JISコードの文字列だもん")

    ex3) Converting to EUC, using auto-detection and converting to zenkaku katakana with
        the full string detection algorithm.
        >>> kc = kconv.Kconv(kconv.SJIS,kconv.AUTO,kconv.ZENKAKU,
                             kconv.FULL)
        >>> print kc.convert("文字コードは任意だよ。")

CUSTOMIZATION
    You can customize kconv by editing defaults.py
        DEFAULT_INPUT_CODING    This will be used then autodetection fails.
        DEFAULT_OUTPUT_CODING   The default output code.
        DEFAULT_BREAKLINE_CODE  Specifies the default breakline character

OTEHR FEATURES
    Kconv contains several other functions for processing Japanese text.
    All functions accept a second argument to specify input encoding just as
    the input encoding in the constructor's argument. The default detection mode is kconv.AUTO.
    The output encoding is EUC.

    *kconv.ChkHiragana(input_string[,icode])
    *kconv.ChkKatakana(input_string[,icode])

    Return 1 if the input string hiragana hiragana(or katakana) charatcters, whitespace,
    breakline character.

    ex)
        >>>print kconv.ChkHiragana("ひらがなだけなんだもん")
        1
        >>>print kconv.ChkKatakana("カタカナだけじゃないみたい")
        0


    *kconv.NumberConvert(input_string[,icode])

    Converts zenkaku numeric characters into hankaku numeric characters.
```

```
    ex)
        >>>print kconv.NumberConvert("342 2 3 6 3 234 3 3 6 8 024")
        '34223632343361024'


    *kconv.Han2Zen(input_string,icode)
    *kconv.Zen2Han(input_stirng,icode)

    Converts hankaku(or zenkaku) characters to zenkaku(or hankaku) characters.
    (This covers roman and non-roman characters)
    Characters that are imposssible to convert will not beconverted.

    ex)
        >>>print kconv.Han2Zen("ぜんかくと Hankaku(^^;")
        ぜんかくとＨａｎｋａｋｕ（^^;
        >>>print kconv.Zen2Han("ぜんかくとＨａｎｋａｋｕ（^^;")
        ぜんかくと Hankaku(^^;


    *kconv.Hira2Kata(input_string,icode)
    *kconv.Kata2Hira(input_string,icode)

    Converts hiragana(or katakana) to katakana(or hiragana).
    Roman letters and numbers are not effected.

    ex)
        >>>print kconv.Hira2Kata("東方は赤く燃えているようです")
        東方ハ赤ク燃エテイルヨウデス
        >>>print kconv.Kata2Hira("デフォルトのコード認識ルーチン")
        でふぉるとのこーど認識るーちん


    *kconv.Upper(input_string,icode)
    *kconv.Lower(input_string,icode)

    Convert lower-case roman letters(or upper-case) to upper-case( or lowercase)
    (on both hankaku and zenkaku)
    Other characters willl not be effected.

    ex)
        >>>print kconv.Upper("Captain ＡＭＥＲＩＣＡ")
        CAPTAIN ＡＭＥＲＩＣＡ
        >>>print kconv.Lower("Hoe Ｕｋｙｕ")
        hoe ｕｋｙｕ


NOTES
    *Kconv is distributed under the most recent GPL.
    *Kconv works with the EUC,JIS,SJIS,Unicode(UCS2),UTF-8 encodings
    *The content of kconv constants are different between the python implementation and
     the C++ version
    *Kconv.FAST is not that fast and sometimes fails to correctly detect the encoding.
```

*Once kconv detects the Unicode encoding in line-by-line mode, it will work under the
 assumption that Unicode is the input encoding for all subsequent lines.
 This is because kconv detects Unicode by using the characters 0x00 and 0xFF
 which are not used in other encodings. All non-Unicode data must not
 include the NULL character 0x00. If it includes the NULL character, kconv
 assume that it is a Unicode string.
*The Python implementation will raise the exception kconv.KconvError when an error occurs.