



# Design and Implementation of a Suite of Chinese Transcoders for Python 2

Tom Emerson (湯姆·愛摩森)  
Senior Computational Linguist

20<sup>th</sup> International Unicode Conference  
Washington, D.C., USA

strategy • process • technology • results

[www.basistech.com](http://www.basistech.com)



## Overview

- What is Python
- Character Sets & Encodings
- Implementation

# What is Python?

- Python is an interpreted, interactive, object-oriented language
  - Classes and introspection
  - Exception mechanism
  - Module system
  - Dynamic Typing
  - Extensive class library
  - Rich Foreign Function interface
  - Unicode strings as first class objects
    - Regular expressions (though not UTR #18)
    - All string methods work on UTF-8 and single-byte strings
    - All string formatting methods work with Unicode strings

# Python's Unicode Support

- Unicode 3.0 fully introduced in Python 2.0
  - Defined by Marc-Andre Lemburg
    - Now documented by PEP 100, *Python Unicode Integration*
- Transcoding supported via the *codecs* module
  - UTF-8<>UCS-2 codecs are built in
    - `uString = u"\uFF11 is fullwidth digit 1"`
    - `uString.encode("UTF-8")`
      - `'\xef\xbc\x91 is fullwidth 1'`
    - `unicode('\xef\xbc\x91 is fullwidth 1', 'UTF-8')`
      - `u'\uff10 is fullwidth digit 1'`
- Unicode properties provided through the *unicodedata* module

# Python's Unicode Support

- 2-byte and 4-byte Unicode characters are supported
  - “\u4E00”
    - ‘\u4e00’
  - “\U0002F9D6”
    - u‘\ud87e\uddd6’
      - Character is stored internally in UTF-16
  - “\U0002F9D6”.encode(“utf-8”)
    - ‘\xf0\xaf\xa7\x96’
      - The Unicode Scalar Value is encoded, *not* the surrogate pair
  - unicode(“\U0002F9D6”.encode(“utf-8”), “utf-8”)
    - u‘\ud87e\uddd6’
- Support for UTF-32 encoded characters proposed in PEP 261, *Support for “Wide” Unicode Characters*.
  - Implemented in Python 2.2
  - Compile time option

## Character Sets vs. Encodings

- Non-Coded Character Sets
  - A non-coded character set represents a list of characters that are defined by an organization as the standard set that one is expected to know.
  - Tōngyòng (7000), Chángyòng (2500), and Cìchángyòng (1000)
- Coded Character Sets
  - A coded character set assigns a unique number (“code point”) to each abstract character in the repertoire.
  - A coded character set does not make any statements about how its code-points are represented on a computer.

## Character Sets vs. Encodings

- An encoding specifies how the code points in a coded character set are to be represented and transmitted with a computer.
- A single character set can have multiple encodings.
- Sometimes the distinction is blurred: Big Five and GB18030-2000 both define a character set and a character encoding.
- Generally laid out in one or more 94x94 grids. Each character is indexed by its row-cell address (qūwèi) within the grid.

## Character Sets

- Simplified
  - GB 2312-80
- Traditional
  - GB 12345-90
  - CNS 11643
  - Big Five, Big Five Plus, ETen
  - GCCS and HKSCS
- “Generic”
  - Unicode/ISO 10646/GB 13000-1992 (Unicode 1.1)
  - GB 18030-2000

# Encodings

- Simplified
  - HZ
    - ASCII, GB 2312:80
  - CN-GB, EUC-CN
    - ASCII, GB 2312:80
  - CP936
    - GBK
- Traditional
  - EUC-TW
    - CNS 11643:1986 Planes 1-2
    - CNS 11643:1992 Planes 3-7
  - Big Five et al.
    - Minimally CNS 11643:1986 1-2 repertoire, HKSCS
  - CP950
    - Big Five + Partial ETen, HKSCS

## Encodings

- GB 18030:2000
  - Unicode 3.x
- ISO 2022-CN
  - ASCII
  - GB 2312:80
  - CNS 11643:1986 Planes 1-2
- ISO 2022-CN-EXT
  - All in 2022-CN
  - ISO-IR-165 (aka CCITT)
    - Includes GB 2312 + extra *hanzi*, Cyrillic, Kana, Bopomofo, and Greek
  - CNS 11643:1992 Planes 3-7

## Commonalities

- Many of these encodings cover the same set of character sets
  - GB 2312:80
    - HZ, CN-GB, EUC-CN, ISO-2022-CN, -EXT
    - CP936 and GB 18030, partially
  - CNS 11643
    - EUC-TW, ISO-2022-CN, -EXT
- We can use a single lookup table for each of these character sets

## But...

- **Big Five and family much more complex**
  - EUDC and VDC areas need to be mapped
    - These are overloaded heavily by HKSCS, Big 5+, CP950, and other extensions
- **CP936/GBK**
  - GB 2312:80 with the rest of the Unicode 1.1 ideographic repertoire
- **GB 18030:2000**
  - GBK with much much more
  - Fortunately mapping tables are not needed for everything

## And Yet...

- ISO 2022 and EUC are “generic”
  - One generic decoder
  - Data driven
  - Extensible for other languages
    - ISO 2022-JP and EUC-JP

## Complications, an Example

Mapping Big 5 UDA2 codepoint 8EA8 to Unicode with various Big 5 variants:

	Big 5	Big 5+	CP950	HKSCS	GB12345	GB2312	Unicode
枂		<b>8EA8</b>					U+67A1
綫		C9A0		<b>8EA8</b>	<u>47-63</u>		U+7DAB
線	<i>BD75</i>	<i>BD75</i>	<i>BD75</i>	<i>BD75</i>			U+7DDA
线						<u>47-63</u>	U+7EBF
			<b>8EA8</b>				PUA

## Goals

- Minimize amount of static data that needs to be carried around.
- Easily updateable mapping tables
- Cross-script/Cross-language reuse

## Encodings

- My hovercraft is full of 鰻魚
  - 鰻 is at 87-9 (0x57 0x09)
  - 魚 is at 51-67 (0x33 0x43)
- My hovercraft is full of 鰻魚
  - 鰻 is at 1-92-13
  - 魚 is at 1-62-03

## Python Codecs

- The codecs module provides a common interface for character transcoding.
- Codecs register themselves with the *codec registry*
  - A codec registers a *search function*
  - Codecs looked up by encoding name
  - Aliasing must be supported by the search function.

## Python Codecs

- A codec must implement four protocols
  - Stateless encoding
  - Stateless decoding
  - Stream (statefull) encoding
  - Stream (statefull) decoding
- Three levels of error handling
  - *Strict*. Raises an exception on error
  - *Ignore*. Ignore the character and continue
  - *Replace*. Substitute with the appropriate replacement character (U+FFFD in Unicode)

## Trivial Codec

```
import codecs, string

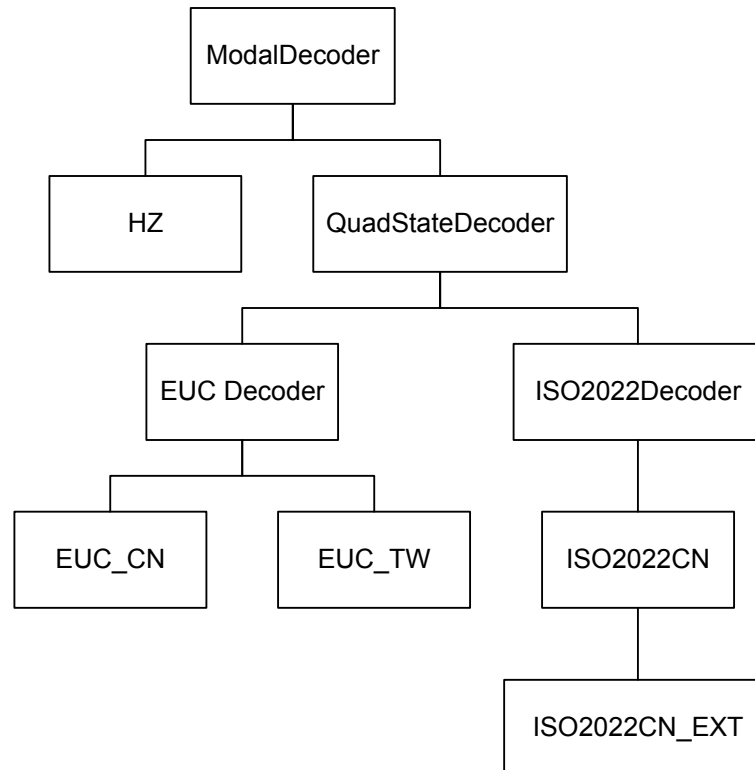
class MyCodec (codecs.Codec):
    def encode(self, input, errors='strict'):
        return (string.upper(input), len(input))

    def decode(self, input, errors='strict'):
        return (string.lower(input), len(input))

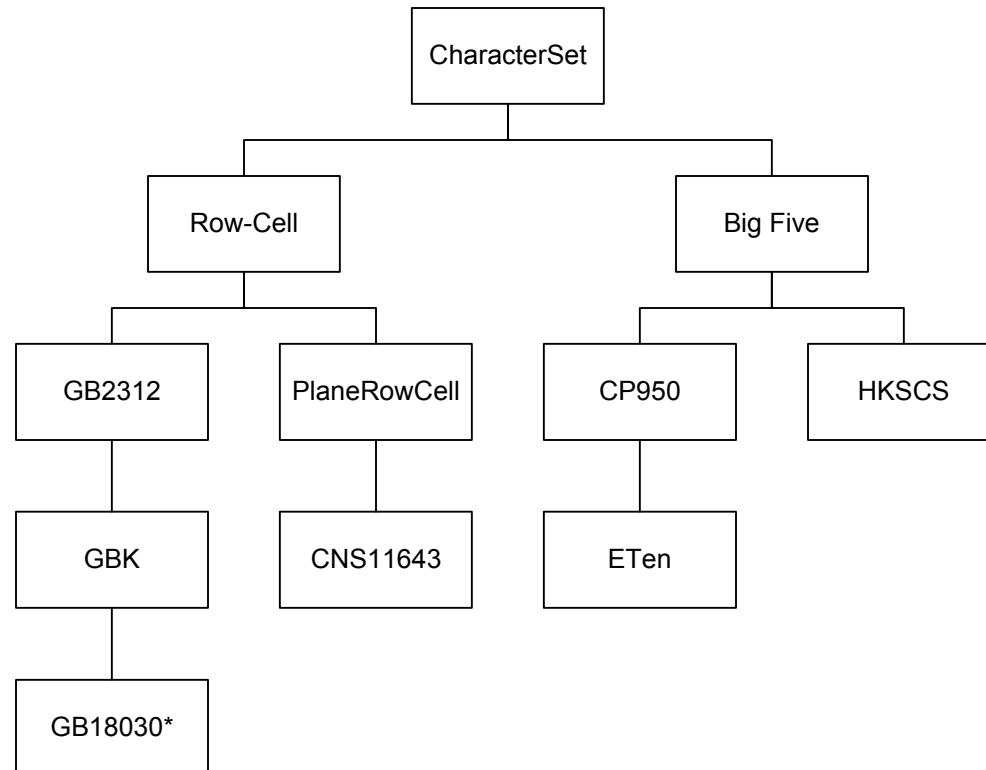
def my_search_function(encoding):
    if encoding == "sample":
        return ( MyCodec().encode, MyCodec().decode,
                codecs.StreamReader, codecs.StreamWriter )

codecs.register(my_search_function)
```

# Decoder Hierarchy



# Character Set Hierarchy



## Implementation Notes

- State machines are key:
  - Needed for ISO 2022, EUC, and HZ
  - Useful for others
- Generic ISO 2022 decoder, supporting 7- and 8-bit versions
  - Based on the ECMA-35 standard
    - Fully identical to 1994 edition of ISO/IEC 2022
  - Generic enough to handle Asian encodings

## Table Representation

- Row-Cell tables are easily represented through nested sequences
  - Indexed via row-cell values, 0-offset
  - Each entry contains the Unicode character (not scalar value) for that location
    - U+FFFD for undefined spots
- Mapping from Unicode to row-cell is done through a mapping (hash) table
  - Returns a sequence of the row-cell location for that Unicode character

## Table Representation

- Tables are generated from raw-data arriving from various sources
  - Obsolete Unicode mapping tables
  - National Standard Organizations
  - Mining from various sources
    - Rosette, ICU, glibc, etc.
- Currently generate Python code
  - One could instead generate C and provide a DLL with the tables.
  - I don't know if this would be faster (yet)
  - Advantageous to allow sharing (via DLLs/SOs)

## Availability

- Basis Technology will be donating this code to the Python community as part of the Python Codecs project hosted at

<http://python-codecs.sourceforge.net>

## Questions

- Final version of these slides will be available at

<http://www.basistech.com/info/>