Python 與中文處理

Tseng Yuen-Hsien，曾元顯

資訊中心

國立臺灣師範大學

2011/10/27

目錄

Python 與中文處理

Python 與中文能否相處得來？

在學習 Python 程式語言處理中文的過程中，碰到一些問題，上網找資料一一解決後，將經驗寫成文字，提供大家參考。

中文編碼：Python 內部表達方式、程式檔案、螢幕輸出

Python 內部表達中文字串時，可以用 Unicode 字串，也可以用 byte string 來儲存與表達中文字串。

使用 Unicode 來表達中文字串，Python 才能在字串的索引中，簡便而正確地取出單一個中文字。例如，在 Python 的命令提示環境下：

```
>>> s=u'中文' # Unicode string
>>> print len(s), s[0] # index 0 fetches the first character
2 中
>>> t='中文' # byte string
>>> print len(t), t[0:3] # a Chinese character has n bytes
6 中
>>> print type(s), type(t)
<type 'unicode'> <type 'str'>
>>> print s, t # this line works fine, 各自輸出
中文 中文
>>> print s + t.decode('utf-8') # 將 t 轉換成 Unicode
中文 中文
```

```
>>> print s + unicode(t, 'utf-8') # 將 t 轉換成 Unicode,效果同上一行
中文 中文
>>> print s.encode('utf-8')+ t # 將 s 轉換成 byte string
中文 中文
>>> print s+t # this will cause an error due to type mismatch
```

將上述 Python 敘述寫在程式檔案中，執行時，要注意到兩件事：

1. 程式檔案的編碼：若存檔成 big5 碼，則要在程式中的第二行，告訴 Python：

```
#!/usr/bin/env python
# -*- coding: big5 -*-
# Note the first line in the above is for Operating system, the
# second line is for Python interpreter
s=u'中文' # big5 code will be stored in Unicode in Python
print len(s), s[0]
```

若程式檔案以 utf-8 編碼方式存檔，則要寫成：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
s=u'中文' # utf-8 code will be stored in Unicode in Python
print len(s), s[0]
t='中文'
print len(t), t[:3]
```

在 utraedit 中檔案的存檔格式也需存為 utf-8，而 cmd 改設定為 chcp 65001

第二行的目的，是要告訴 Python 解譯器，請用該編碼方式處理後面的所有字串常數。如上例第二行指定用 utf-8，則第三行 s=u'中文'裡面的「中文」兩個字，會被 Python 認為是以 utf-8 編碼的，然後轉成內部的中文 Unicode（字串常數前面加個 u）或是 byte string。若你**在 Mac（或是 Unix）環境中，以 utf-8 儲存程式檔案，傳輸到 Windows 中修改，然後儲存成 big5 碼，這時第二行也要跟著修改成 big5**。若存檔時的編碼方式，跟第二行指定的**編碼方式不同，則 Python 執行該程式檔案時就會出錯**。

2. 輸出視窗的編碼：Windows 的命令提示字元視窗（即 DOS 視窗），是以 big5 方式顯示中文。而 Mac（或是 Unix）的終端機，則可以選擇 big5 或是 utf-8 等編碼顯示中文字。以 Python 敘述將中文字串輸出到螢幕時，若要看到正常的中文字（而不是亂碼），則要先知道輸出視窗的編碼，將輸出字串編碼後輸出，才能正確顯示中文字。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
s=u'中文' # utf-8 code will be stored in Unicode in Python
print s.encode('utf-8') # assume output screen is utf-8
t='中文'
print t # no need for encode(), assume output screen is utf-8
```

若在 Mac（或是 Unix）中執行程式，輸出正常的中文字，但在 Windows 中輸出有誤，則可檢查輸出的敘述，其編碼方式是否符合輸出視窗的編碼，反之亦然。

註：這真是麻煩，能否不修改程式，而能在不同的輸出視窗中，正確輸出中文字呢？若輸出到瀏覽器，就沒有這個問題（瀏覽器可以自動或手動調整編碼，正確顯示）

註：字串是 Unicode 時，有時候直接 print 出來，也可以看到正常的中文字。猜想是 Python 的 print 能對 Unicode 自動做編碼，將字串轉換過再輸出。

中文編碼：輸入檔案、輸出檔案

了解前述說明後，在 Python 程式中，將中文字串寫到檔案時，只要指定其編碼、輸出字串、關閉檔案，再以文字編輯器，用正確的編碼方式打開，即可看到正常的中文字。例如：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
f = open('Big5File.txt', 'w')
for word in ['中文', 'Python', u'處理', u'訣竅']:
  if isinstance(word, unicode): # u'處理' and u'訣竅' are Unicode
    f.write(word.encode('big5'))
  else: # '中文' is in utf-8, converted into Unicode by decode()
    f.write( (word.decode('utf-8')).encode('big5') )
f.close()
```

在 Python 程式中，讀入中文文字檔案時，也要事先知道該檔案的編碼方式，才能正確讀取。例如：

```
import codecs
f = codecs.open('Big5File.txt', 'rb', 'big5', 'replace')
for line in f:
  print line.encode('utf-8') # this works if screen is utf-8
  print line # will this work fine, too?
```

上面範例中的參數 replace：也可以改成 ignore 等，其效果如下：

```
>>> unicode('\x80abc', errors='replace') # convert into Unicode
u'\ufffdabc'
>>> unicode('\x80abc', errors='ignore') # convert into Unicode
u'abc'
>>> unicode('\x80abc', errors='strict')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeDecodeError: 'ascii' codec can't decode byte 0x80 in
position 0: ordinal not in range(128)
```

因為預設（沒有指定時）是 strict，要使用 replace 或是 ignore 才能避免讀檔錯誤。

什麼時候用 encode()、什麼時候用 decode()

將 big5、utf-8 等碼轉成 Unicode 的動作（將外面的中文字串轉成 Python 內部的 Unicode 字串）時，則呼叫 decode()函數，反向的動作，則呼叫 encode()函數。

如何知道某一個字串（或是文字檔案）的編碼？

Python 有一個模組：chardet。安裝完後，可以這樣用（參見參考書目）：

```
import chardet
rawdata = open(infile, "r").read()
result = chardet.detect(rawdata)
charenc = result['encoding']
inF=open(infile,"rb")
s=unicode(inF.read(),charenc)
inF.close()
```

Unicode、utf-8、utf-16、utf-32

Unicode 不等於 utf-8 或是 utf-16，看看下面的說明，可了解其差異。

Unicode 是對每一個「字」（中文字、日文字、英文字母、俄文字母、標點符號等）用一個數字來代表。至於這個數字儲存在電腦中的方法，就叫做編碼方法。Big5、utf-8、utf-16 等等，都是屬於一種編碼方法。底下的例子，來自於參考資料：「Characters vs. Bytes」。

下面四個字的 Unicode 碼如下：

```
U+0026 AMPERSAND (decimal 38)
U+0416 CYRILLIC CAPITAL LETTER ZHE (decimal 1,046)
U+4E2D HAN IDEOGRAPH 4E2D (decimal 20,013)
U+10346 GOTHIC LETTER FAIHU (decimal 66,374)
```

將他們用 utf-8 編碼，其通則如下： 以二進位來看，第一個位元組（byte）的前面幾個位元（bit）代表該字是由幾個 bytes 組成；其後 bytes 的最前面兩個位元均為 10。

U+0026 AMPERSAND 需要一個 byte 即可，其 utf-8 編碼為 0010 1100_2=0x26（十進位是 38）。請注意紅色位元的部份。

U+0416 的二進位是 0100 0001 1100_2，其 utf-8 編碼為

1101 0000_2 1001 1100_2=0xD0 0x96

用到兩個 bytes，請注意紅色位元，以及底色位元的部份。

U+4E2D 的二進位是 0100 1110 0010 1101_2，其 utf-8 編碼為

1110 0100 1011 1000_2 1010 1101_2=0xE4 0xB8 0xAD

用到三個 bytes。

U+10346 的二進位是 001 0000 0011 0100 0110_2，其 utf-8 編碼為

1111 0000 1001 0000 1000 1101_2 1000 0110_2=0xF0 0x90 0x8D 0x86

用到四個 bytes。

因此，utf-8 編碼有時候是一個位元組（byte）代表一個字，有時候二個位元組、三個位元組、或是四個位元組，視該字在 Unicode 中所代表的數字而定。

至於 utf-16 的編碼，跟 utf-8 的方式類似，最少兩個位元組（大部分的字都如此），少部分的字，用到四個位元組。因此，若文字檔都是英文文字，用 utf-16 編碼會浪費一半的記憶空間。

最後，utf-32 的編碼，則將 Unicode 的每個字都用四個位元組表示，最浪費空間，看起來最省事（因為不用像 utf-8 處理位元層次的編碼），但 CPU 的計算速度比記憶體的存取速度快，因此，碼的轉換，不見得比較有效率。

參考書目

Characters vs. Bytes, http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF. Updated: 2003/04/26

Character detection in a text file in Python using the Universal Encoding Detector (chardet), http://stackoverflow.com/questions/3323770/character-detection-in-a-text-file-in-python-using-the-universal-encoding-detect.