Lesson 13: Handling Unicode

Fundamentals of Text Processing for Linguists

Na-Rae Han

Objectives

- Shameek's presentation:
  - Object-oriented programming
- Handling Unicode

The ASCII chart

- http://en.cppreference.com/w/cpp/language/ascii
- http://web.alfredstate.edu/weimandn/miscellaneous/ascii/ASCII%20Conversion%20Chart.pdf

Decimal	Binary (7-bit)	Character
0	000 0000	(NULL)
…	…	…
35	010 0011	#
36	010 0100	&
…	…	…
48	011 0000	0
49	011 0001	1
50	011 0010	2
…	…	…

Decimal	Binary (7-bit)	Character
65	100 0001	A
66	100 0010	B
67	100 0011	C
…	…	…
97	110 0001	a
98	110 0010	b
99	110 0011	c
…	…	…
127	111 1111	(DEL)

Extending ASCII: ISO-8859, etc.

- ASCII (=7 bit, 128 characters) was sufficient for encoding English. But what about characters used in other languages?
- Solution: Extend ASCII into 8-bit (=256 characters) and use the additional 128 slots for non-English characters
  - **ISO-8859**: has 16 different implementations!
    - ISO-8859-1 aka Latin-1: French, German, Spanish, etc.
    - ISO-8859-7 Greek alphabet
    - ISO-8859-8 Hebrew alphabet
  - JIS X 0208: Japanese characters

🡐 Problem: overlapping character code space.

224_{dec} means **à** in Latin-1 but **א** in ISO-8859-8!

Unicode

- A character encoding standard developed by the Unicode Consortium
- Provides a single representation for *all* world's writing systems
- "Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.”

  (http://www.unicode.org)

How big is Unicode?

- Version 6.2 (2012) has codes for 110,182 characters
  - Full Unicode standard uses **32 bits** (**4 bytes**) : it can represent 2^{32} = 4,294,967,296 characters!

    ← In reality, only 21 bits are needed

- Unicode has three encoding versions
  - UTF-32 (32 bits/4 bytes): direct representation
  - UTF-16 (16 bits/2 bytes): 2^{16}=65,536 possibilities
  - UTF-8 (8 bits/1 byte): 2^8=256 possibilities

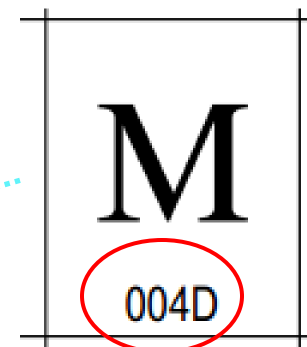
  - Why UTF-16 and UTF-8?
    - They are more compact (for certain languages, i.e., English)

A look at Unicode chart

- How to find your Unicode character:
  - http://www.unicode.org/standard/where/
  - http://www.unicode.org/charts/

- Basic Latin (ASCII)
  - http://www.unicode.org/charts/PDF/U0000.pdf

<table>
<tr><th></th><th>000</th><th>001</th><th>002</th><th>003</th><th>004</th><th>005</th><th>006</th><th>007</th></tr>
<tr><td>0</td><td>NUL
0000</td><td>DLE
0010</td><td>SP
0020</td><td>0
0030</td><td>@
0040</td><td>P
0050</td><td>`
0060</td><td>p
0070</td></tr>
<tr><td>1</td><td>SOH
0001</td><td>DC1
0011</td><td>!
0021</td><td>1
0031</td><td>A
0041</td><td>Q
0051</td><td>a
0061</td><td>q
0071</td></tr>
<tr><td>2</td><td>STX
0002</td><td>DC2
0012</td><td>"
0022</td><td>2
0032</td><td>B
0042</td><td>R
0052</td><td>b
0062</td><td>r
0072</td></tr>
<tr><td>3</td><td>ETX
0003</td><td>DC3
0013</td><td>#
0023</td><td>3
0033</td><td>C
0043</td><td>S
0053</td><td>c
0063</td><td>s
0073</td></tr>
<tr><td>4</td><td>EOT</td><td>DC4</td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td></tr>
<tr><td>C</td><td>FF
000C</td><td>FS
001C</td><td>,
002C</td><td><
003C</td><td>L
004C</td><td>\
005C</td><td>l
006C</td><td>|
007C</td></tr>
<tr><td>D</td><td>CR
000D</td><td>GS
001D</td><td>-
002D</td><td>=
003D</td><td>M
004D</td><td>]
005D</td><td>m
006D</td><td>}
007D</td></tr>
<tr><td>E</td><td>SO
000E</td><td>RS
001E</td><td>.
002E</td><td>>
003E</td><td>N
004E</td><td>^
005E</td><td>n
006E</td><td>~
007E</td></tr>
<tr><td>F</td><td>SI
000F</td><td>US
001F</td><td>/
002F</td><td>?
003F</td><td>O
004F</td><td>_
005F</td><td>o
006F</td><td>DEL
007F</td></tr>
</table>



Code point for *M*.
But "**004D**"?

Another representation: hexadecimal

Hexadecimal (hex) = base-16

- Utilizes 16 characters:

  0 1 2 3 4 5 6 7 8 9 A B C D E F

- Designed for human readability & easy byte conversion
  - 2^4=16: 1 hexadecimal digit is equivalent to 4 bits
  - 1 byte (=8 bits) is encoded with just 2 hex chars!

Letter	Base-10 (decimal)	Base-2 (binary)	Base-16 (hex)
M	77	0000 0000 0100 1101	**004D**

  - Unicode characters are usually referenced by their hexadecimal code
  - Lower-number characters go by their 4-char hex codes, e.g. **U+004D** ("M", U+ for Unicode)
  - Higher-number characters go by 5 or 6 hex codes, e.g. **U+1D122** (http://www.unicode.org/charts/PDF/U1D100.pdf)

Code point representation in Python

Escape sequence	Meaning	Example
\uxxxx	Unicode character with 16-bit hex value xxxx	u'\u004D'
\Uxxxxxxxx	Unicode character with 32-bit hex value xxxxxxxx	u'\U0000004D'
\xhh	Character with hex value hh	'\x4D'

```
>>> print u'\u004D'           # Unicode, 16-bit (utf-16)
M
>>> print u'\U0000004D'       # Unicode, 32-bit (utf-32)
M
>>> print '\x4D'              # ordinary string in hex
M
>>> type(u'\u004D')
<type 'unicode'>
>>> type('\x4D')
<type 'str'>
```

Unicode strings are of type **unicode**, different from **str**

Unicode vs. str type

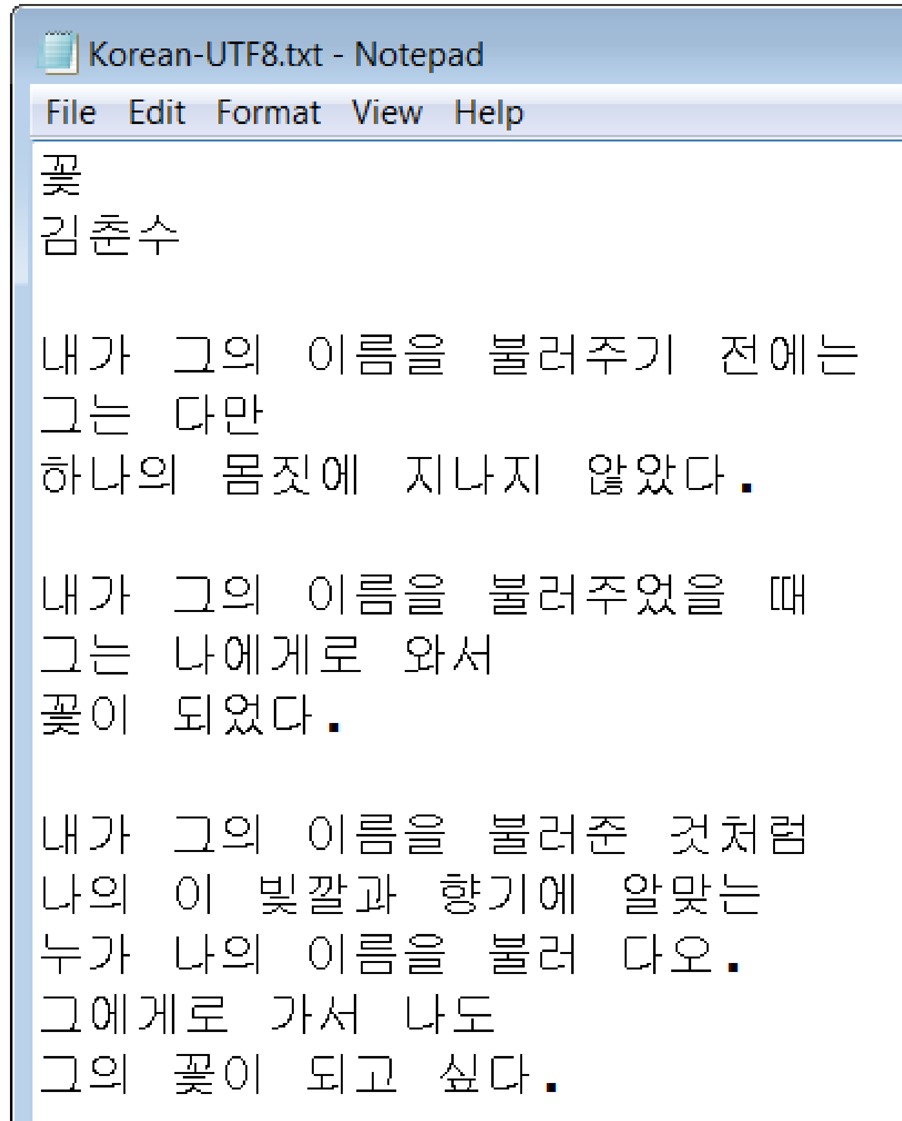
```
>>> type(u'\u004D')
<type 'unicode'>
>>> type('\x4D')
<type 'str'>

>>> unicode('M')
u'M'
>>> 'M'.decode()
u'M'
>>> u'M'.encode()
'M'
```

Unicode strings are of type **unicode**, different from **str**

unicode <-> str conversion

A file with Korean unicode text



Read unicode (or non-ascii) files

- Use codecs module

```
>>> import codecs
>>> f = codecs.open('Korean-UTF8.txt', encoding='utf-8')
>>> lines = f.readlines()
>>> f.close()
>>> lines[0]
u'\uaf43 \r\n'
>>> print lines[0],
꽃
>>> lines[5]
u'\ud558\ub098\uc758 \ubab8\uc9d3\uc5d0 \uc9c0\ub098\uc9c0
\uc54a\uc558\ub2e4. \r\n'
>>> print lines[5],
하나의 몸짓에 지나지 않았다.
>>>
```

codecs.open() let you specify encoding of a file

Internally represented as unicode, prints out fine using system font

Write to unicode (or non-ascii) files

- Again, use `codecs.open()` when opening a file for writing

```
>>> f = codecs.open('lala.txt', 'w', encoding='euc_kr')
>>> f.write(lines[5])
>>> f.close()
>>>
```

Use **`codecs.open()`** when opening a file object for writing

utf-8? euc_kr?

- Encoding names can be found on this page:
  - Python's standard encodings

  https://docs.python.org/2/library/codecs.html#standard-encodings

- Finding the right encoding for a file can be tricky.
  - Encoding information is often available on the text editor you use.
  - Windows: Notepad++ is highly recommended.
    - http://notepad-plus-plus.org/

Wrap-up

- **Next class**
  - Looking forward: NLTK (Natural Language Toolkit)

- **No exercise this week!**
  - But try out unicode on your own