Lesson 13: Handling Unicode
Objectives

- Shameek's presentation:
  - Object-oriented programming

- Handling Unicode
## The ASCII chart


<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (7-bit)</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000 0000</td>
<td>(NULL)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>35</td>
<td>010 0011</td>
<td>#</td>
</tr>
<tr>
<td>36</td>
<td>010 0100</td>
<td>&amp;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>48</td>
<td>011 0000</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>011 0001</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>011 0010</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (7-bit)</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>100 0001</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>100 0010</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>100 0011</td>
<td>C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>97</td>
<td>110 0001</td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td>110 0010</td>
<td>b</td>
</tr>
<tr>
<td>99</td>
<td>110 0011</td>
<td>c</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>127</td>
<td>111 1111</td>
<td>(DEL)</td>
</tr>
</tbody>
</table>
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.

\[ 224_{\text{dec}} \text{ means } \text{à in Latin-1 but } \text{א in ISO-8859-8!} \]
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)

4/9/2014
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)
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!

<table>
<thead>
<tr>
<th>Letter</th>
<th>Base-10 (decimal)</th>
<th>Base-2 (binary)</th>
<th>Base-16 (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>77</td>
<td>0000 0000 0100 1101</td>
<td>004D</td>
</tr>
</tbody>
</table>

- Unicode characters are usually referenced by their hexadecimal code
- Lower-number characters go by their 4-char hex codes, e.g. \texttt{U+004D} ("M", U+ for Unicode)
- Higher-number characters go by 5 or 6 hex codes, e.g. \texttt{U+1D122} ([http://www.unicode.org/charts/PDF/U1D100.pdf](http://www.unicode.org/charts/PDF/U1D100.pdf))
## Code point representation in Python

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\uxxxxx</td>
<td>Unicode character with 16-bit hex value xxxx</td>
<td>u'\u004D'</td>
</tr>
<tr>
<td>\Uxxxxxxxxxxx</td>
<td>Unicode character with 32-bit hex value xxxxxxxx</td>
<td>u'\U0000004D'</td>
</tr>
<tr>
<td>\xhh</td>
<td>Character with hex value hh</td>
<td>'\x4D'</td>
</tr>
</tbody>
</table>

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

Unicode strings are of type `unicode`, different from `str`
Unicode vs. str type

```python
>>> 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
꽃
김춘수

내가 그의 이름을 불러주기 전에는
그는 다만
하나의 몸짓에 지나지 않았다.

내가 그의 이름을 불러주었을 때
그는 나에게로 와서
꽃이 되었다.

내가 그의 이름을 불러준 것처럼
나의 이 빛깔과 향기에 알맞는
누가 나의 이름을 불러 다오.
그에게로 가서 나도
그의 꽃이 되고 싶다.
Read unicode (or non-ascii) files

- Use `codecs` module

```python
>>> import codecs
>>> f = codecs.open('Korean-UTF8.txt', encoding='utf-8')
>>> lines = f.readlines()
>>> f.close()
>>> lines[0]
u'\uf43 \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.
>>> 
```
Write to unicode (or non-ascii) files

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

```python
>>> 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](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.
Wrap-up

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

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