Advanced Python II

by Raymond Hettinger
@raymondh

All about Unicode

```
This is a string:
Löwis луч
LÖWIS ЛУЧ   <-- upper-cased

It has 9 characters:
L 76 LATIN CAPITAL LETTER L which is in Lu
ö 246 LATIN SMALL LETTER O WITH DIAERESIS which is in Ll
w 119 LATIN SMALL LETTER W which is in Ll
i 105 LATIN SMALL LETTER I which is in Ll
s 115 LATIN SMALL LETTER S which is in Ll
  32 SPACE which is in Zs
л 1083 CYRILLIC SMALL LETTER EL which is in Ll
у 1091 CYRILLIC SMALL LETTER U which is in Ll
ч 1095 CYRILLIC SMALL LETTER CHE which is in Ll
```

Number properties

```
>>> from unicodedata import category, name
>>> n = '\u0664\u0662’
>>> for c in n:
        print(c, name(c), category(c))

٤ ARABIC-INDIC DIGIT FOUR Nd
٢ ARABIC-INDIC DIGIT TWO Nd

>>> print(n)
٤٢
>>> print(int(n))
42
```

Combining Characters

```
>>> t = 'L' + chr(111) + chr(776) + 'wis'
>>> print(t)
Löwis
>>> len(t)
6

>>> s = normalize('NFC', t)
>>> print(s)
Löwis
>>> print(len(s))
5
>>> print([ord(c) for c in s])
[76, 246, 119, 105, 115]
```

Implication of properties and normalization

- There are equivalence classes (many ways to represent the same glyph and multiple glyphs with a single meaning)
- Database lookup needs to be normalized consistently
- Numbers need to be converted to actual numeric values

Bidirectional Text

- For scripts like Arabic and Hebrew which are written right-to-left
- However numbers are written left-to-right
- There are direction markers, Unicode Control Characters, to switch modes:
  - U+200E Left-to-right Mark
  - U+200F Right-to-left Mark
- There are a number of interesting wrinkles, collectively referred to as Bidi control marks including strong and weak marks for main direction and temporary reversals.

Encodings

- These code points:

[76, 246, 119, 105, 115, 32, 1083, 1091, 1095]

Have many possible encodings:

ascii [76, 63, 119, 105, 115, 32, 63, 63, 63]
latin-1 [76, 246, 119, 105, 115, 32, 63, 63, 63]
iso-8859-5 [76, 63, 119, 105, 115, 32, 219, 227, 231]
utf-8 [76, 195, 182, 119, 105, 115, 32, 208, 187, 209, 131, 209, 135]
utf-16_be [0, 76, 0, 246, 0, 119, 0, 105, 0, 115, 0, 32, 4, 59, 4, 67, 4, 71]

Properties of Encodings

- Single byte versus multi-byte
- Fixed width versus variable length
- Self-synchronizing
- Detectable
- Space efficiency
- Covers all of unicode versus just a subset
- Random strings may or may not have meaning

Two basic strategies for encoding

Only encode a subset of Unicode in sequence of bytes

- ASCII — Code points U+0000 to U+007F
- Latin-1 — Code points U+0000 to U+00FF
- Latin/Cyrillic — ASCII plus U+0451 to U+045F

Encode all of Unicode in multibyte sequences

- UTF-8
- UTF-16
- UTF-32

UTF-8 and UTF-16

```
c = 1095


print('utf-8’,
                c//64+129+64,
                c%64+128)

print('utf-16’,
                c//256, c%256)
```

UTF-16 beyond the BMP

- The c//256 and c%256 works only up to 65535
- Beyond that, it takes two code units (totaling 4 bytes) for each character (code point)
- Those are called surrogate pairs
- They are distinct which makes them self-synchronizing

Hack of the Day (binary data in JSON)

- JSON is UTF-8 by definition
- UTF-8 is not valid for binary data
- But, Latin-1 is!
- With further ado:

```
data.decode('latin-1')
```

- That makes a unicode string out of binary data
- Which can then be encoded in UTF-8 by a JSON encoder

How do you know the encoding?

HTTP Header:

Content-Type: text/html; charset=iso-8859-1

MIME Type:

Mime-Version: 1.0
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit

Standard encoding for a given operating system (MCBS, etc)

Detecting Encodings

- Guess UTF-8 first
  - Very low odds of incorrect acceptance
  - Always works for ASCII
- Unreliably, do pattern matching for letter frequencies (electronic equivalent of trying common encodings and looking for mojibake)

Discuss common encodings

- Those code points:

[76, 246, 119, 105, 115, 32, 1083, 1091, 1095]

Have many possible encodings:

ascii [76, 63, 119, 105, 115, 32, 63, 63, 63]
latin-1 [76, 246, 119, 105, 115, 32, 63, 63, 63]
iso-8859-5 [76, 63, 119, 105, 115, 32, 219, 227, 231]
utf-8 [76, 195, 182, 119, 105, 115, 32, 208, 187, 209, 131, 209, 135]
utf-16_be [0, 76, 0, 246, 0, 119, 0, 105, 0, 115, 0, 32, 4, 59, 4, 67, 4, 71]

Goal check

- Learn difference between text and bytes
- Know differences between common encodings
- Don’t confuse encode() and decode()
- Vocabulary:
  - Code unit
  - Code point
  - Surrogate Pair
  - Combining Pair
  - Normalization
  - Byte order marks
  - Equivalence classes
  - Code properties
  - Bidi control marks

A word on WSGI

- The basic problem is how to recover input text from an unknown encoding
- Read the whatsnew3.2 write-up
- Discuss