#--------------------------------------------------------------------------

# Examples related to the ..\unicodemod.py script.

#

# \*\* unicodemod.py avoids the issues illustrated here by always

# \*\* discarding Unicode BOM code points in input data retained by

# \*\* a 'from' encoding name. This file illustrates the BOM behavior

# \*\* of Python encoding names, and pertains to more general usage.

#

# To support encoding conversions for file data, you must use an input

# encoding name that strips the Unicode BOM (byte order marker) used in

# some encodings, if one is present at the start of the file. Otherwise,

#

# 1) A retained BOM's code point in the file's data string may not

# convert to other narrower encodings such as Latin-1, even if all

# the file's actual text is compatible with the new encoding.

#

# 2) A retained BOM may be written to the output file redundantly when

# converting to encodings that add BOMs of their own (e.g., UTF-16).

#

# To drop the BOM:

#

# -For UTF-8 files: use 'from' encoding "utf-8-sig" \*in all cases\*.

# This encoding discards the BOM only if one is present, but "utf-8"

# does not. For output, use either "utf-8-sig" or "utf-8"; the former

# adds a BOM, and the latter does not.

#

# -For UTF-16 files: use 'from' encoding "utf-16" if a BOM is present,

# and the order-specific encodings "utf-16-le" or "utf-16-be" otherwise.

# Unlike UTF-8, there is no predefined UTF-16 input encoding that always

# discards the BOM only if one is present. Order-specific encodings

# like "utf-16-le" do not add the BOM on output, but "utf-16" does.

#

# -For UTF-32 files: the rules are the same for UTF-16 (but replace

# "16" in encoding names with "32").

#

# The UTF-8/UTF-16 difference seems asymmetric, but it's per the Unicode

# standard (UTF-16 BOMs are to be omitted for uses such as database fields).

# For more details, see "Learning Python, 5th Edition", and the official

# word at http://unicode.org/faq/utf\_bom.html#BOM.

#--------------------------------------------------------------------------

#==========================================================================

# UTF-8:

# -for output, utf-8-sig writes BOM, utf-8 does not;

# -for input, utf-8-sig skips BOM iff present, but utf-8 retains it;

# You should generally always use this utf-8-sig for input; because

# utf-8 does not skip a BOM if present, byte 0 in its read results

# will fail to convert to other encodings (e.g., latin-1).

#==========================================================================

>>> open('utf-8.txt', 'w', encoding='utf-8').write('SPAM')

>>> open('utf-8-sig.txt', 'w', encoding='utf-8-sig').write('SPAM')

>>> open('utf-8.txt', 'rb').read()

b'SPAM'

>>> open('utf-8-sig.txt', 'rb').read()

b'\xef\xbb\xbfSPAM'

>>> open('utf-8.txt', 'r', encoding='utf-8-sig').read() # sig skips bom iff present

'SPAM'

>>> open('utf-8-sig.txt', 'r', encoding='utf-8-sig').read()

'SPAM'

>>> open('utf-8.txt', 'r', encoding='utf-8').read()

'SPAM'

>>> open('utf-8-sig.txt', 'r', encoding='utf-8').read() # non-sig retains bom if present

'\ufeffSPAM' # <= won't encode to Latin-1!

#==========================================================================

# UTF-16:

# -for output, utf-16 writes BOM, utf-16-le does not;

# -for input, utf-16 both requires and skips BOM, utf-16-le does neither;

# Python could write a BOM automatically for utf-16-le, but per a 2007

# dev issue report, the Unicode standard FAQ requires the BOM be omitted.

#==========================================================================

>>> open('utf-16.txt', 'w', encoding='utf-16').write('SPAM')

>>> open('utf-16-le.txt', 'w', encoding='utf-16-le').write('SPAM')

>>> open('utf-16.txt', 'rb').read()

b'\xff\xfeS\x00P\x00A\x00M\x00'

>>> open('utf-16-le.txt', 'rb').read()

b'S\x00P\x00A\x00M\x00'

>>> open('utf-16.txt', 'r', encoding='utf-16').read() # skips and requires bom

'SPAM'

>>> open('utf-16-le.txt', 'r', encoding='utf-16').read()

UnicodeError: UTF-16 stream does not start with BOM

>>> open('utf-16.txt', 'r', encoding='utf-16-le').read() # le retains bom if present

'\ufeffSPAM' # <= won't encode to Latin-1!

>>> open('utf-16-le.txt', 'r', encoding='utf-16-le').read()

'SPAM'

#==========================================================================

# UTF-32:

# -for output, utf-16 writes BOM, utf-16-le does not;

# -for input, utf-16 both requires and skips BOM, utf-16-le does neither;

# Same as UTF-16, but replace "16" with "32" in encoding names.

#==========================================================================

>>> open('utf-32.txt', 'w', encoding='utf-32').write('SPAM')

>>> open('utf-32-le.txt', 'w', encoding='utf-32-le').write('SPAM')

>>> open('utf-32.txt', 'rb').read()

b'\xff\xfe\x00\x00S\x00\x00\x00P\x00\x00\x00A\x00\x00\x00M\x00\x00\x00'

>>> open('utf-32-le.txt', 'rb').read()

b'S\x00\x00\x00P\x00\x00\x00A\x00\x00\x00M\x00\x00\x00'

>>> open('utf-32.txt', 'r', encoding='utf-32').read() # skips and requires bom

'SPAM'

>>> open('utf-32-le.txt', 'r', encoding='utf-32').read()

UnicodeError: UTF-32 stream does not start with BOM

>>> open('utf-32.txt', 'r', encoding='utf-32-le').read() # le retains bom if present

'\ufeffSPAM' # <= won't encode to Latin-1!

>>> open('utf-32-le.txt', 'r', encoding='utf-32-le').read()

'SPAM'