Unicode and bytes

All Versions

Examples

Encoding/decoding error handling

.encode and .decode both have error modes.

The default is 'strict' , which raises exceptions on error. Other modes are more forgiving.

Encoding

```
>>> "£13.55".encode('ascii', errors='replace')
b'?13.55'
>>> "£13.55".encode('ascii', errors='ignore')
b'13.55'
>>> "£13.55".encode('ascii', errors='namereplace')
b'\\N{POUND SIGN}13.55'
>>> "£13.55".encode('ascii', errors='xmlcharrefreplace')
b'&#163;13.55'
>>> "£13.55".encode('ascii', errors='backslashreplace')
b'\\xa313.55'
```

Decoding

```
>>> b = "£13.55".encode('utf8')
>>> b.decode('ascii', errors='replace')
'��13.55'
>>> b.decode('ascii', errors='ignore')
'13.55'
>>> b.decode('ascii', errors='backslashreplace')
'\\xc2\\xa313.55'
```

Morale

It is clear from the above that it is vital to keep your encodings straight when dealing with unicode and bytes.

File I/O

Files opened in a non-binary mode (e.g. 'r' or 'w') deal with strings. The deafult encoding is 'utf8' .

```
open(fn, mode='r')                    # opens file for reading in utf8
open(fn, mode='r', encoding='utf16')  # opens file for reading utf16

# ERROR: cannot write bytes when a string is expected:
open("foo.txt", "w").write(b"foo")
```

Files opened in a binary mode (e.g. 'rb' or 'wb') deal with bytes. No encoding argument can be specified as there is no encoding.

```
open(fn, mode='wb')  # open file for writing bytes

# ERROR: cannot write string when bytes is expected:
open(fn, mode='wb').write("hi")
```

Basics

In Python 3 str is the type for unicode-enabled strings, while bytes is the type for sequences of raw bytes.

```
type("f") == type(u"f")  # True, <class 'str'>
type(b"f")               # <class 'bytes'>
```

In Python 2 a casual string was a sequence of raw bytes by default and the unicode string was every string with "u" prefix.

```
type("f") == type(b"f")  # True, <type 'str'>
type(u"f")               # <type 'unicode'>
```

Unicode to bytes

Unicode strings can be converted to bytes with .encode(encoding) .

Python 3

```
>>> "£13.55".encode('utf8')
b'\xc2\xa313.55'
>>> "£13.55".encode('utf16')
b'\xff\xfe\xa3\x001\x003\x00.\x005\x005\x00'
```

Python 2

in py2 the default console encoding is sys.getdefaultencoding() == 'ascii' and not utf-8 as in py3, therefore printing it as in the previous example is not directly possible.

```
>>> print type(u"£13.55".encode('utf8'))
<type 'str'>
>>> print u"£13.55".encode('utf8')
SyntaxError: Non-ASCII character '\xc2' in...

# with encoding set inside a file

# -*- coding: utf-8 -*-
>>> print u"£13.55".encode('utf8')
┬ú13.55
```

If the encoding can't handle the string, a `UnicodeEncodeError` is raised:

```
>>> "£13.55".encode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character '\xa3' in position 0: ordinal not in ra
```

Bytes to unicode

Bytes can be converted to unicode strings with .decode(encoding) .

A sequence of bytes can only be converted into a unicode string via the appropriate encoding!

```
>>> b'\xc2\xa313.55'.decode('utf8')
'£13.55'
```

If the encoding can't handle the string, a UnicodeDecodeError is raised:

```
>>> b'\xc2\xa313.55'.decode('utf16')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/csaftoiu/csaftoiu-github/yahoo-groups-backup/.virtualenv/bin/../lib/python3.5/enc
    return codecs.utf_16_decode(input, errors, True)
UnicodeDecodeError: 'utf-16-le' codec can't decode byte 0x35 in position 6: truncated data
```

Syntax

```
str.encode(encoding, errors='strict')
```

```
bytes.decode(encoding, errors='strict')
```

```
open(filename, mode, encoding=None)
```

Parameters

Parameter	Details
encoding	The encoding to use, e.g. 'ascii' , 'utf8' , etc...
errors	The errors mode, e.g. 'replace' to replace bad characters with question marks, 'ignore' to ignore bad characters, etc...

Remarks