python and unicode

pycon italia

antonio cavedoni

python and unicode

Don’t be scared!

This stuff is easy

So what is text (in the computer sense)?

a sequence of characters

a'ight: which characters?
how are they stored?

Different ways of storing data on disk

ASCII (7-bit + high bit code pages for european languages) American Standard Code for Information Interchange

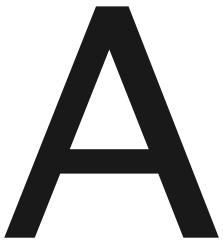
Multibyte code pages, like Shift-JIS, which uses the bit range 0x80-0xff to denote the start of a multibyte sequence of chars

Unicode, which is **not** an encoding but also has a family of encodings

Unicode

Each glyph is represented by a “code point”, which is an abstract entity.

U+0041
LATIN CAPITAL LETTER A

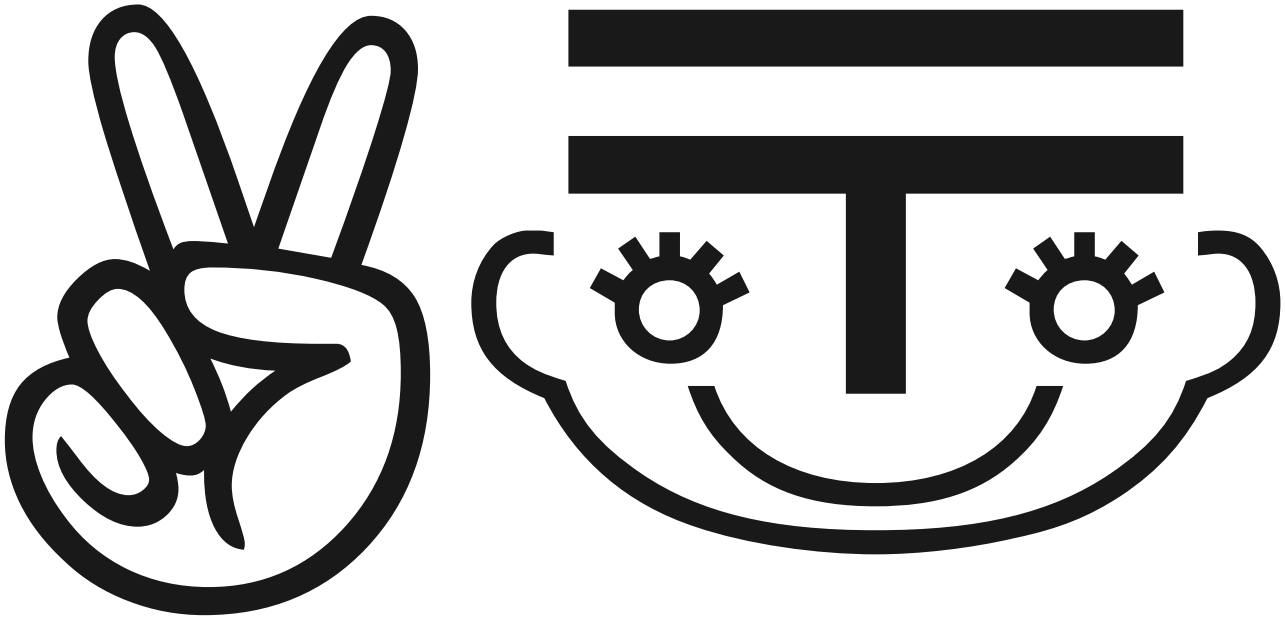


U+03A3
GREEK CAPITAL LETTER SIGMA

Favourites

U+270C
VICTORY HAND

U+3020
POSTAL MARK FACE



Favourites

U+FFFD

REPLACEMENT CHARACTER

Encodings

Code points don’t necessary relate to the way they are actually stored on-disk. That’s what Unicode **encodings** are for. There are several kinds of encodings.

Encodings

UTF-8 which is a multibyte encoding: ASCII characters are preserved as-is, all the other code points get the multibyte treatment

UTF-16 (also known as UCS-2) and UTF-32 (UCS-4) where each code point is stored in either combinations of 2 or 4 bytes.

Joel's article on Unicode

Popular article by Joel Spolsky, which you should check out.

http://www.joelonsoftware.com/articles/Unicode.html

There ain’t no such thing as plain text

If you don’t know the encoding of a string, all bets are off.

To actually do something with a string you absolutely need to know how it’s encoded, otherwise it’s just a hopeless struggle. You can try to do some guesswork and there are packages that help with it.

Ugly guesswork

This is what web browsers do. There is a Python implementation of this strategy that is a direct port of how Mozilla handles encoding problems. It's by Mark Pilgrim, feedparser author, and is called Universal Encoding Detector.

http://chardet.feedparser.org/

python and unicode

Python Unicode support

Python's Unicode support is very good and has been for a while

...will get even better in Python 3000 when Unicode strings will be the default

How to get a unicode string

Several ways

By typing it in:

```
>>> foo = u"Hello, world."
>>> foo = u"A letter: \u0041"
```

By decoding an existing byte string:

```
>>> foo = "Accented a: à".decode("utf-8")
>>> foo = unicode("Accented a: à", "utf-8")
```

unicode() on strings

Accepts two additional parameters

encoding specifies the codec to decode the string

error may be ‘strict’ (raises ValueError on errors, default), ‘ignore’ (causes errors to be silently ignored) or ‘replace’ (substitutes undecodable characters with U+FFFD, REPLACEMENT CHARACTER)

unicode() calls _unicode_()

Return the Unicode string version of object

For objects which provide a _unicode_ method, it will call this method without arguments to create a Unicode string. For all other objects, the 8-bit string version or representation is requested and then converted to a Unicode string using the codec for the default encoding in strict mode.

decode()

Strings have decode() methods that you can use to get Unicode objects, similar to the unicode() built-in

Calling by name

You can also call a Unicode code point with other escape sequences

by name

```
>>> foo = u"\N{POSTAL MARK FACE}"
```

by code point below U+FFFF

```
>>> foo = u"\u00f1"
```

U+00F1 LATIN SMALL LETTER N WITH TILDE

by code point above U+FFFF

```
>>> foo = u"\U00010346"
```

U+10346 GOTHIC LETTER FAIHU

Converting encodings

Example: converting a string from one encoding to another

(assuming foo is a latin1 string gathered from somewhere)

```
>>> foo = foo.decode("latin1").encode("utf-8")
```

Source code files

You can specify the encoding of each Python code file with a comment on the first line (after the python interpreter one)

```
# -*- coding: utf-8 -*-

import os, sys
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os, sys
```

Source code files

What this enables is to use UTF-8 bytestrings directly inside source file strings

```
# -*- coding: utf-8 -*-
foo = "Accented a: à"
foo = foo.decode("utf-8")
```

python and unicode

common pitfalls

Printing: UnicodeEncodeError

You are trying to printing a Unicode string and nothing works

```
>>> print u"\N{POSTAL MARK FACE}"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeEncodeError: "ascii" codec can't
encode character u"\u3020" in position 0:
ordinal not in range(128)
```

Printing: UnicodeEncodeError

You are trying to printing a Unicode string and nothing works

Because the default codec is ascii. Change it something actually able to encode that character, like UTF-8 and if your terminal supports it, you should see

```
>>> print u"\N{POSTAL MARK FACE}".\
encode("utf-8")
〒
```

Can’t decode: UnicodeDecodeError

The reverse is true as well: you’re trying to get a Unicode object from a bytestring but it doesn’t work

```
>>> unicode("chaîne de caractères", "ascii")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeDecodeError: ‘ascii’ codec can’t
decode byte 0xee in position 3: ordinal not in
range(128)
```

Can’t decode: UnicodeDecodeError

This happens when you try to create a Unicode object but the codec can’t decode it, or an encoding is not specified for unicode() and the default (sys.getdefaultencoding()) gets used

Can’t decode

Also happens when you try to encode an already-encoded bytestring

```
>>> face_utf8 = u"\N{POSTAL MARK FACE}".\
encode("utf-8")
>>> print face_utf8
〒
>>> face_utf8.encode("utf-8")
Traceback (most recent call last):
  File “<stdin>”, line 1, in ?
UnicodeDecodeError: ‘ascii’ codec can’t
decode byte 0xe3 in position 0: ordinal not in
range(128)
```

Unicode{Encode|Decode}Error: not evil!

UnicodeDecodeError and UnicodeEncodeError are **not evil** and are in fact a way for Python to try and get you to ask yourself these questions

Where does this string come from? Is it encoded? How? In which piece of code?

Unicode{Encode|Decode}Error: not evil!

Besides, it's probably not even a Unicode problem, it's more a problem with the reality of how text is stored on a computer

python and unicode

stdlib modules

codecs

important functions provided by this module are open, which lets you open an encoded file on-disk and read it (or write) as Unicode transparently, and register, which allows writing write custom codec handlers. It also has some constants for BOMs, etc.

unicodedata

unicodedata has the name function that given a unicode character can return its name according to the Unicode database file (3.2 until Python 2.4, 4.1.0 since Python 2.5), lookup function that gives you a Unicode character given its name and normalize function which handles NFC/NFD, NFKC/NFKD conversions.

unicodedata can also tell you in which unicode group does a character belong, if it's a digit or not and its bidirectional category

re

Python regular expressions can search Unicode strings too, just pass them the re.UNICODE or re.U parameter

python and unicode

thank you