Introduction to Human Language Technologies

Tomaž Erjavec

Karl-Franzens-Universität Graz

Lecture: Character sets
11.1.2008

Overview

1. **Basic concepts**
2. **ASCII**
3. **8-bit character sets**
4. **Unicode**
5. **Python**

Computer coding of characters

- Computers store data as (binary) numbers
- There is no a priori relationship between these numbers and characters (of an alphabet)
- If there are no conventions for mapping numbers to characters, or there are too many conventions --> chaos
- Standards and quasi standards:
  ASCII, ISO 8859, (Windows, Mac), Unicode

Basic concepts I.

- *a character*
  - an abstract concept
    (An „A“ is something like a Platonic entity: it is the idea of an „A“ and not the „A“ itself)
  - of itself a character does not have a mapping to a number of a concrete visual representation
  - so, characters are usu. defined descriptively, e.g. „Greek small letter alpha“; the graphical representation is given only as an examplar, „α“

Basic concepts II.

- *character repertoire* or *coded character set*
  - a set of characters
  - each character is associated with a number (a *character code*)
  - identical characters can belong to different characters sets if they are logicaly distinct, e.g. capital letter A in the Latin alphabet, in the Cyrillic alphabet, capital alpha in Greek
- *character code (codepoint)*
  - a 1-1 relation between the character from a character set and a number e.g.
    A = 26, B = 27, ...

Basic Concepts III.

- *character encoding*
  - an algorithm, which translates the character code into a concerete digital encoding, in bytes
- *byte / octet*
  - the minimal unit that is processed by a computer
  - typically 8 bits (0/1) : 0-255

Basic concepts IV

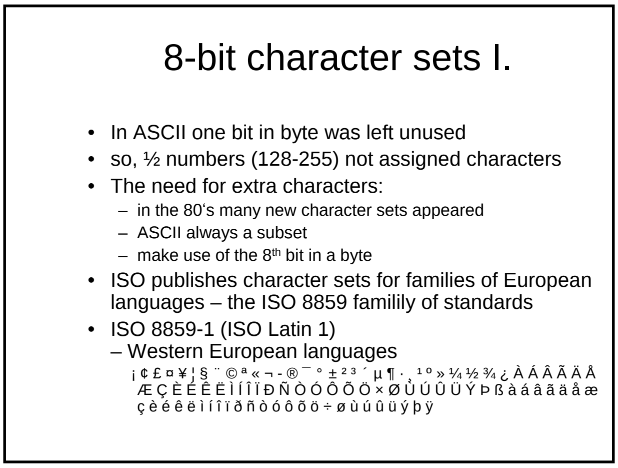
- *glyph*
  - the graphical representation of a character
  - a character can have several glyphs: A, 𝒜, 𝕬
  - sometimes one glyph can have several characters, e.g. the glyph “P” corresponds to the Latin letter P, the Cyrillic letter Er or Greek Rho
- *font*
  - the graphical representations of a set of characters for some character repertoire (coded character set):
    𝕬, B, C, Č, D, ...

ASCII

- American Standard Code for Information Interchange (1950’)
- a 7-bit character set: range from 0-127
- 0-31 - control codes and formatting:
  Escape, Line Feed, Tab, Space,...
- 32-126 – punctuation etc., numbers, lc and English letters :
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
  @ A B C D E F G H I J K L M N O P Q R S T U V W
  X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v
  w x y z { | } ~
- Oxymoron: ‘8-bit ASCII’

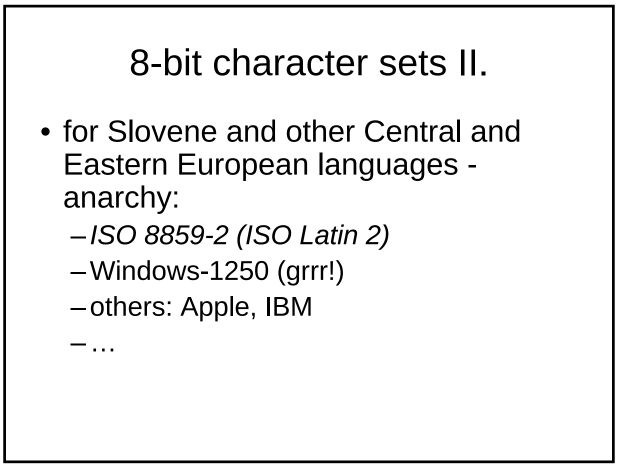
ASCII II.

- Advantages:
  - no chaos:
    one character - one codepoint (number)
  - trivial character encoding algorithm:
    one codepoint - one byte
- Weakness:
  - does not support non-English characters



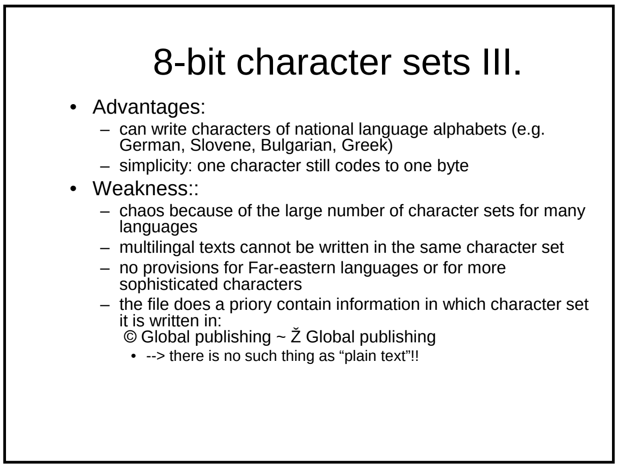
8-bit character sets I.

- In ASCII one bit in byte was left unused
- so, ½ numbers (128-255) not assigned characters
- The need for extra characters:
  - in the 80's many new character sets appeared
  - ASCII always a subset
  - make use of the 8th bit in a byte
- ISO publishes character sets for families of European languages – the ISO 8859 famlily of standards
- ISO 8859-1 (ISO Latin 1)
  – Western European languages
  ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ



8-bit character sets II.

- for Slovene and other Central and Eastern European languages - anarchy:
  - *ISO 8859-2 (ISO Latin 2)*
  - Windows-1250 (grrr!)
  - others: Apple, IBM
  - …



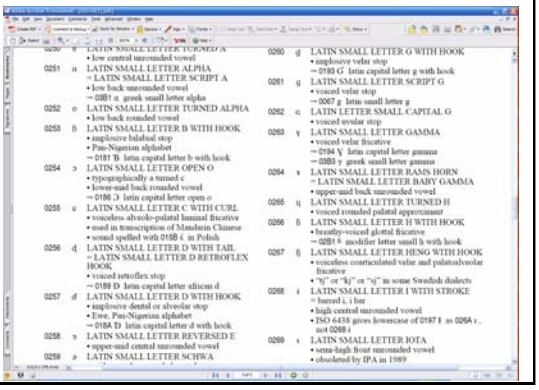
8-bit character sets III.

- Advantages:
  - can write characters of national language alphabets (e.g. German, Slovene, Bulgarian, Greek)
  - simplicity: one character still codes to one byte
- Weakness::
  - chaos because of the large number of character sets for many languages
  - multilingal texts cannot be written in the same character set
  - no provisions for Far-eastern languages or for more sophisticated characters
  - the file does a priory contain information in which character set it is written in:
    © Global publishing ~ Ž Global publishing
    - --> there is no such thing as “plain text”!!

Unicode I.

- If we want to extend the character set, the only solution is to code one character in several bytes
- 1991 – Unicode Consortium: http://www.unicode.org/
- *ISO 10646 Unicode*
  - defines the universal character set
  - defines 30 alphabets covering several hundred languages, cca 40.000 characterov
  - …CJK, Arabic, Sanskrt,…
  - historical alphabets, punctuation, math symbols, diacritics, …
  - A character definition in Unicode:
„LATIN CAPITAL LETTER A WITH ACUTE“

Unicode definitions for IPA



0250 ɐ LATIN SMALL LETTER TURNED A
• low central unrounded vowel
0251 ɑ LATIN SMALL LETTER ALPHA
= LATIN SMALL LETTER SCRIPT A
• low back unrounded vowel
→ 03B1 α greek small letter alpha
0252 ɒ LATIN SMALL LETTER TURNED ALPHA
• low back rounded vowel
0253 ɓ LATIN SMALL LETTER B WITH HOOK
• implosive bilabial stop
• Pan-Nigerian alphabet
→ 0181 Ɓ latin capital letter b with hook
0254 ɔ LATIN SMALL LETTER OPEN O
• typographically a turned c
• lower-mid back rounded vowel
→ 0186 Ɔ latin capital letter open o
0255 ɕ LATIN SMALL LETTER C WITH CURL
• voiceless alveolo-palatal laminal fricative
• used in transcription of Mandarin Chinese
• sound spelled with 015B ś in Polish
0256 ɖ LATIN SMALL LETTER D WITH TAIL
= LATIN SMALL LETTER D RETROFLEX HOOK
• voiced retroflex stop
→ 0189 Ɖ latin capital letter african d
0257 ɗ LATIN SMALL LETTER D WITH HOOK
• implosive dental or alveolar stop
• Ewe, Pan-Nigerian alphabet
→ 018A Ɗ latin capital letter d with hook
0258 ɘ LATIN SMALL LETTER REVERSED E
• upper-mid central unrounded vowel
0259 ə LATIN SMALL LETTER SCHWA

0260 ɠ LATIN SMALL LETTER G WITH HOOK
• implosive velar stop
→ 0193 Ɠ latin capital letter g with hook
0261 ɡ LATIN SMALL LETTER SCRIPT G
• voiced velar stop
→ 0067 g latin small letter g
0262 ɢ LATIN LETTER SMALL CAPITAL G
• voiced uvular stop
0263 ɣ LATIN SMALL LETTER GAMMA
• voiced velar fricative
→ 0194 Ɣ latin capital letter gamma
→ 03B3 γ greek small letter gamma
0264 ɤ LATIN SMALL LETTER RAMS HORN
= LATIN SMALL LETTER BABY GAMMA
• upper-mid back unrounded vowel
0265 ɥ LATIN SMALL LETTER TURNED H
• voiced rounded palatal approximant
0266 ɦ LATIN SMALL LETTER H WITH HOOK
• breathy-voiced glottal fricative
→ 02B1 ʱ modifier letter small h with hook
0267 ɧ LATIN SMALL LETTER HENG WITH HOOK
• voiceless coarticulated velar and palatoalveolar fricative
• "tj" or "kj" or "sj" in some Swedish dialects
0268 ɨ LATIN SMALL LETTER I WITH STROKE
= barred i, i bar
• high central unrounded vowel
• ISO 6438 gives lowercase of 0197 Ɨ as 026A ɪ, not 0268 ɨ
0269 ɩ LATIN SMALL LETTER IOTA
• semi-high front unrounded vowel
• obsoleted by IPA in 1989

Unicode II.

- 1 character ≠ 1 bye, what now?
- for Unicode, several character encodings exist:
  - UTF-32
    - 1 character – 4 bytes
  - UTF-16
    - if BMP character (Basic Multilingual Plane)
1 character – 2 bytes
    - otherwise
1 character – 4 bytes
  - **UTF-8**
    - varying length: 1-6 bytes for character
    - if character in ASCII then one byte (compatibility)
    - most European characters code in two bytes

Unicode III.

- diactrics exists as zero width characters (combining diacritical marks)
- e.g. a +ˆ + ̤ = â̤
- but problems with displaying complex combinations,
- e.g. a +ˆ + ˚ = â

Back to ASCII

ASCII is sometimes still the only safe encoding:

- how to keyboard complex characters
- how to transfer text (e-mail, www)

Re-coding to ASCII:

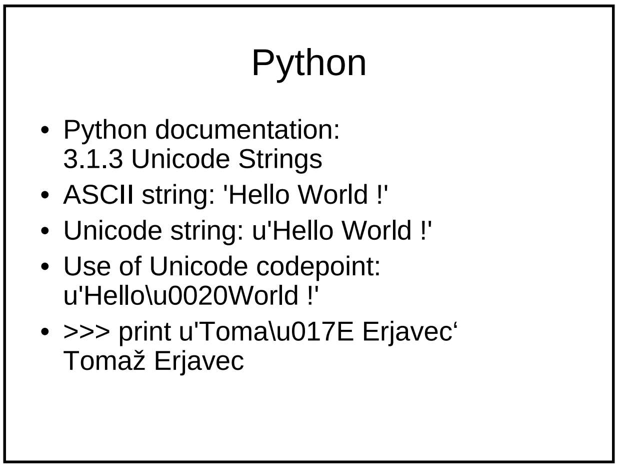
- ■ e-mail - MIME standard
- ■ WWW - Unidoce character entities, e.g. &#353; (= &#x160;) = š

Conversion between character sets

- Linux:

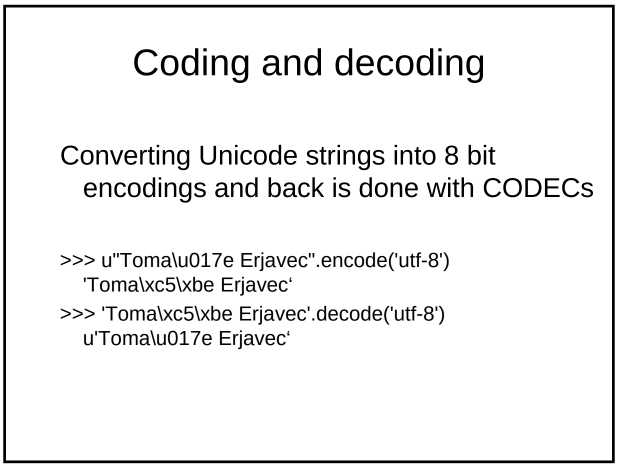
```
iconv –f windows-1250 –t utf8 text-win > text-utf8
```

- Windows:
  - charmap
  - MS Word / Save as



Python

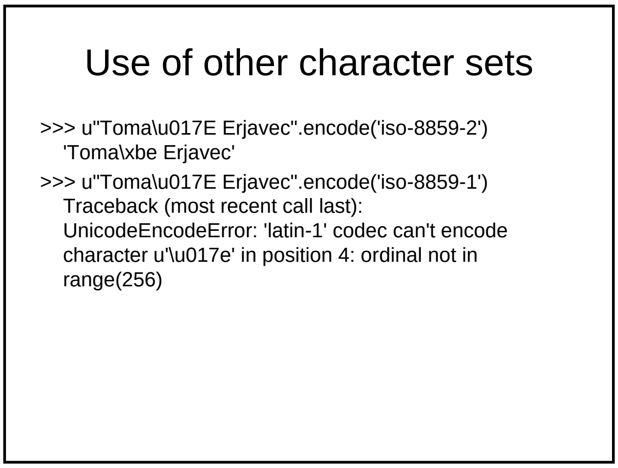
- Python documentation:
  3.1.3 Unicode Strings
- ASCII string: 'Hello World !'
- Unicode string: u'Hello World !'
- Use of Unicode codepoint:
  u'Hello\u0020World !'
- >>> print u'Toma\u017E Erjavec‘
  Tomaž Erjavec



Coding and decoding

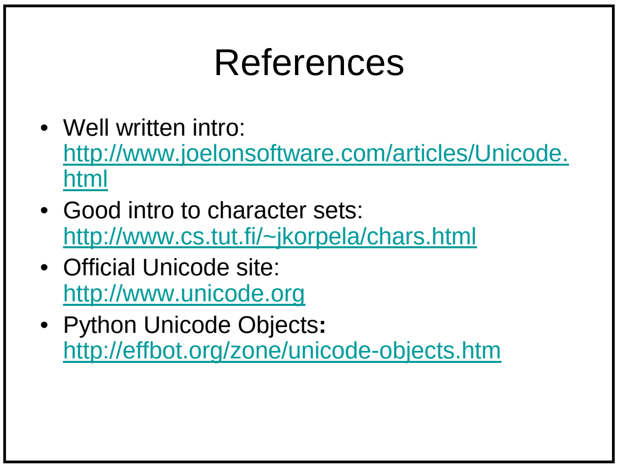
Converting Unicode strings into 8 bit encodings and back is done with CODECs

```
>>> u"Toma\u017e Erjavec".encode('utf-8')
    'Toma\xc5\xbe Erjavec‘
>>> 'Toma\xc5\xbe Erjavec'.decode('utf-8')
    u'Toma\u017e Erjavec‘
```



Use of other character sets

```
>>> u"Toma\u017E Erjavec".encode('iso-8859-2')
    'Toma\xbe Erjavec'
>>> u"Toma\u017E Erjavec".encode('iso-8859-1')
    Traceback (most recent call last):
    UnicodeEncodeError: 'latin-1' codec can't encode
    character u'\u017e' in position 4: ordinal not in
    range(256)
```



References

- Well written intro: http://www.joelonsoftware.com/articles/Unicode.html
- Good intro to character sets: http://www.cs.tut.fi/~jkorpela/chars.html
- Official Unicode site: http://www.unicode.org
- Python Unicode Objects**:** http://effbot.org/zone/unicode-objects.htm