ZEICHENSÄTZE, ZEICHENSATZKODIERUNG, UNICODE UND KODIERUNGSKONVERSION

MASTERSEMINAR SUCHMASCHINEN UND RAG

STEFAN LANGER, CIS, UNIVERSITÄT MÜNCHEN

SOMMERSEMESTER 2024

Motivation

Dokumente, die von Suchmaschinen verarbeitet werden, können in verschiedensten Formaten auftreten

Der Text kann in verschiedensten Zeichensatzkodierungen vorliegen

Die interne Verarbeitung in einer Suchmaschine sollte in einer festgelegten Zeichensatzkodierung erfolgen

Zeichensätze

Was ist ein Zeichensatz, eine Zeichensatzkodierung ...

- Terminologie
- Einige Beispiele

Unicode

- UTF-8
- UTF-16, UTF-32 u.a. Unicode-Kodierungen

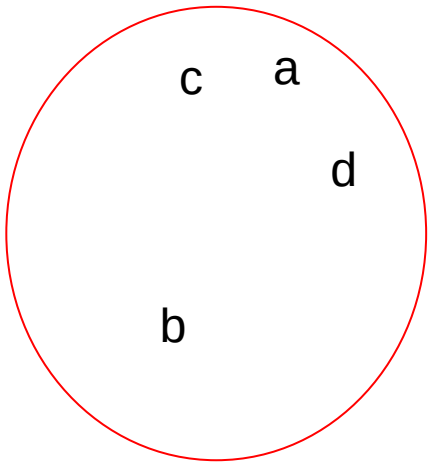
Andere Zeichensätze mit mehr als 256 Zeichen

Hebräisch, Arabisch (Schreibrichtung rechts nach links)

Kodierungskonversion

Zeichensatzerkennung

Terminologie

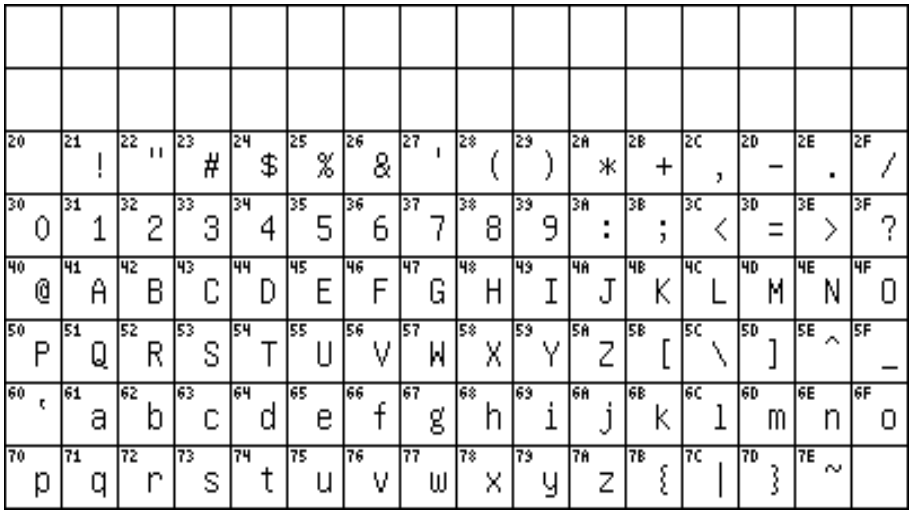
Zeichensatz character set character repertoire	**nummerierter Zeichensatz** coded character set	**Zeichensatzkodierung** character encoding (scheme)
	a → 1 b → 2 c → 3 d → 4	a → 1 → 00000001 b → 2 → 00000010 c → 3 → 00000011 d →4 → 00000100

ASCII

ASCII

ASCII definiert nur den Bereich 0-127

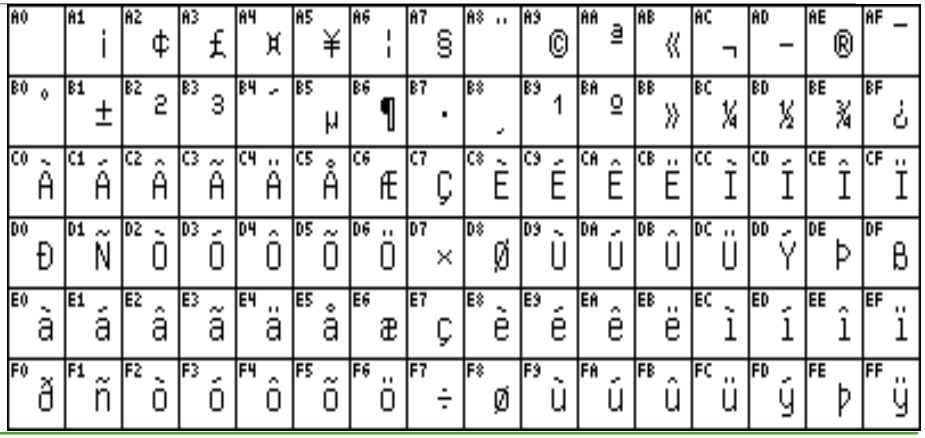
0-31 ist mit Kontrollzeichen besetzt



ISO-8859-1 versus Windows-1252

Iso-8859-1 (=Iso-Latin-1)

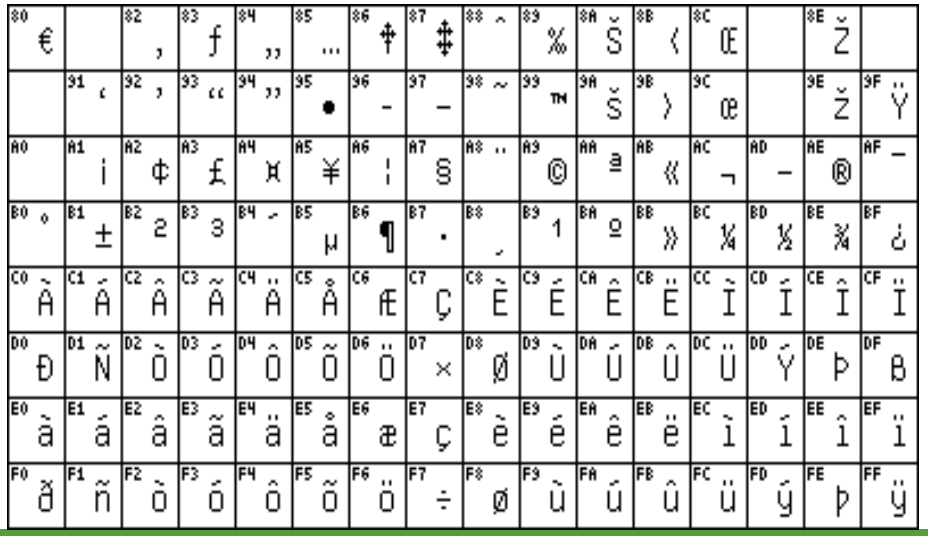
Iso-8859-X Zeichensätze benutzen nicht den Bereich x80 bis x9F



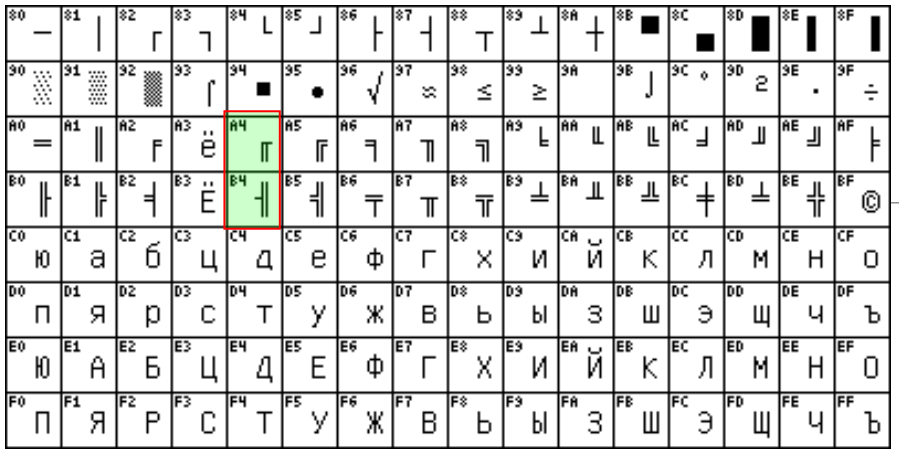
Windows-1252

Bereich x80-x9F benutzt

Sonst identisch zu ISO-8859-1

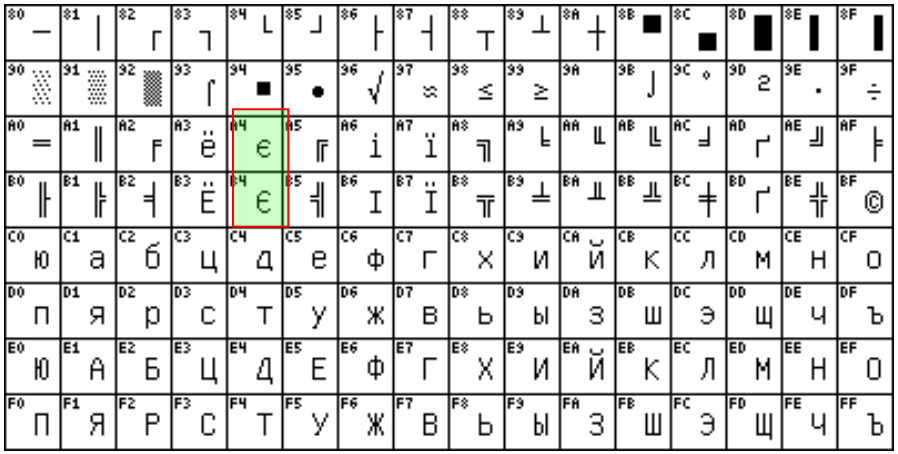


Kyrillische Zeichensätze

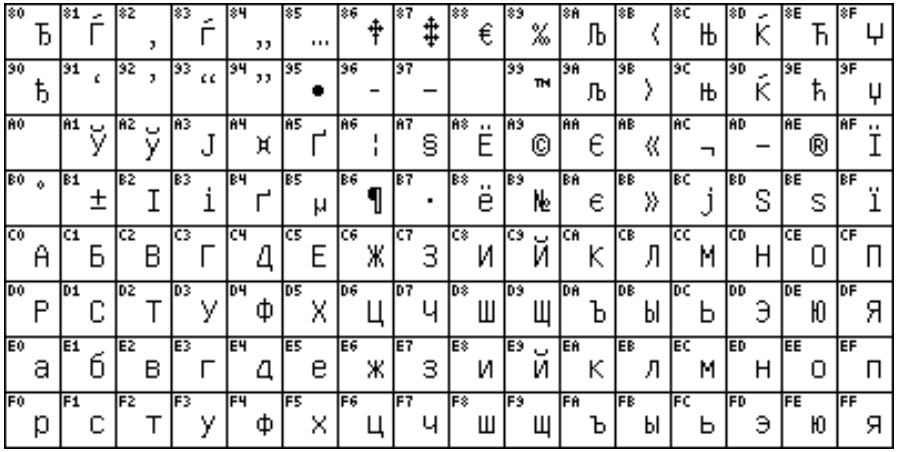


KOI8-R - Russisch

ISO-8859-5



KOI8-U - Ukrainisch



Windows-1251

Für den die kyrillische Schrift existieren zahlreiche unterschiedliche Zeichensätze, die z.T. alle, z.T. nur den für eine Sprache relevanten Teil der kyrillischen Buchstaben enthalten.

Zeichensatzkodierungen in einem Byte

Alle Zeichensätze mit <= 256 Zeichen und einer entsprechenden Nummerierung 0-255 (< 2^8) können in einem Byte 1 zu 1 kodiert werden.

Der numerische Wert des Bytes ist gleich der der Kodeposition des Zeichens im nummerierten Zeichensatz

Wenn der Zeichensatz ASCII-kompatibel sein soll, sind nur 128 Positionen zusätzlich zu den 128 mit ASCII-Zeichen besetzten Positionen verfügbar (z.B. in Windows 1252).

ISO-8859-X-Zeichensätze haben nur 96 frei besetzbare Kodepositionen, da die Positionen 0-127 gleich wie ASCII ist, und die Positionen 128-160 nicht besetzt sind.

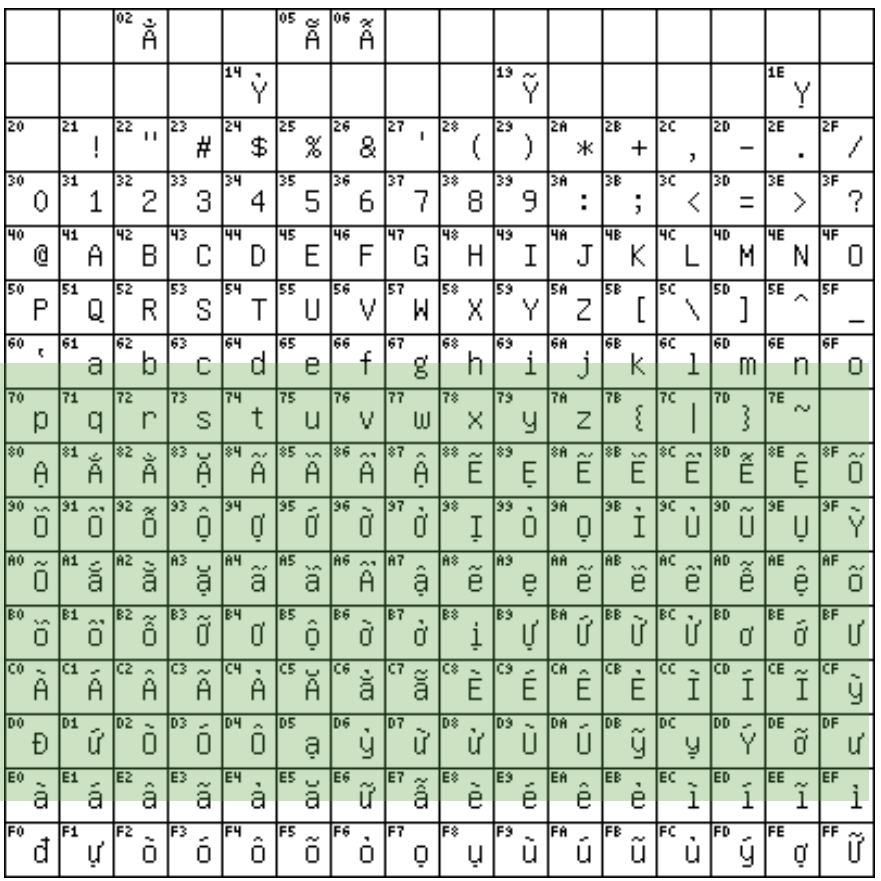
VISCII – die Grenzen des Bytes

Vietnamese – VISCII

Mehr als 128 zusätzliche Buchstaben; daher muss der Bereich < 128 (<x80) geändert werden →

6 zusätzliche Buchstaben im Bereich 0-31

Rest von ASCII ist unverändert



Übungsaufgabe (5 min)

Die vorgestellten Methoden zur Zeichensatzkodierungen sind nicht geeignet für manche Sprachen – welche?

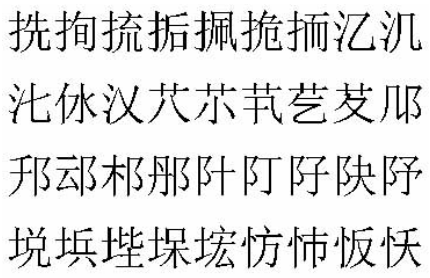
Welche Lösungen können Sie sich für solche Zeichensätze vorstellen

Große Zeichensätze

Größere Zeichensätze

CJK – Chinesisch, Japanisch, Koreanisch – haben wesentlich mehr als 256 Zeichen.

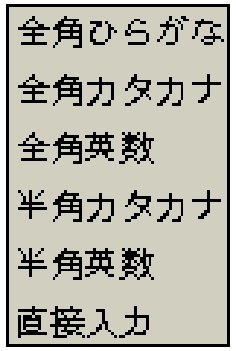
Japanisch verwendet neben Hiragana und Katakana (Silbenschrift) auch Zeichen chinesischen Ursprungs (Kanji); in Koreanisch werden neben Hangul (s.u.) auch noch chinesische Schriftzeichen (Hanja) verwendet.



Chinesisch **Japanisch**

(Kanji+Hiragana/Katakana)

Koreanisch (Hangul)

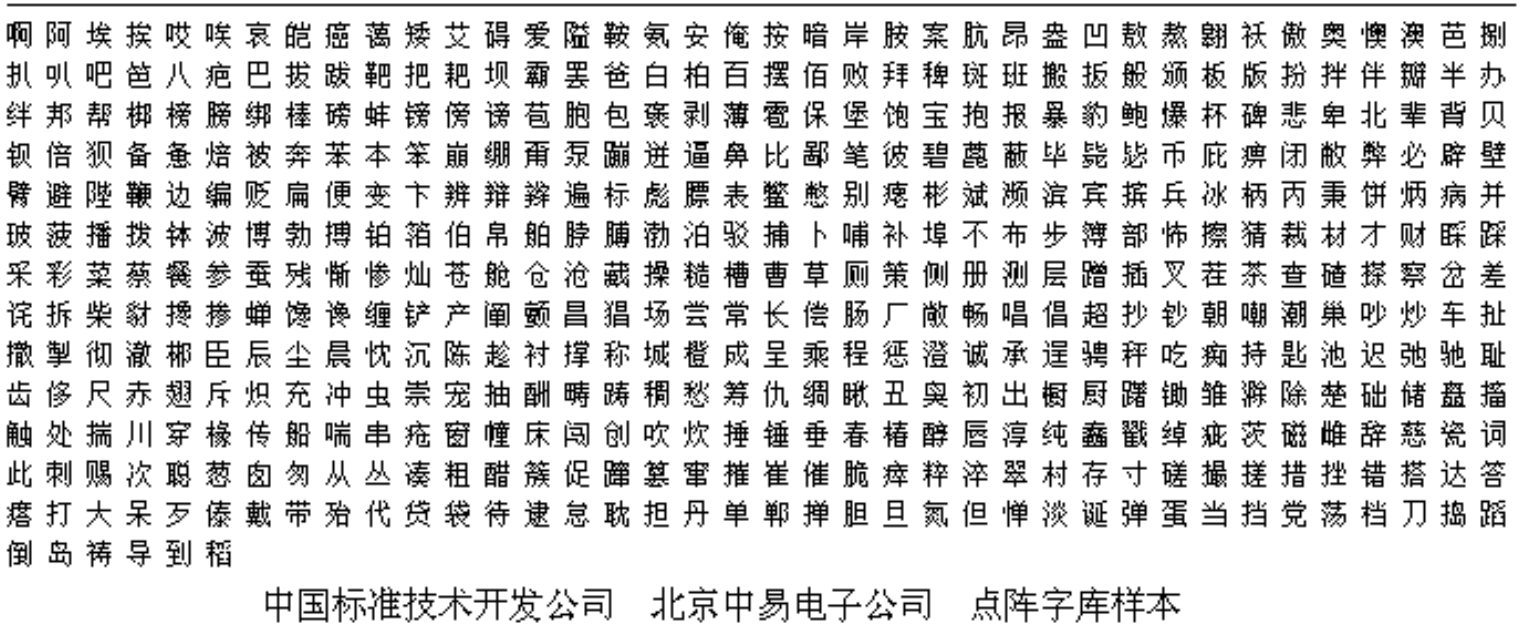


모든 인간은 태어날 때부 터 자유로우며 그 존엄과 권리에 있어 동등하다. 인간은 천부적 으로 이성과 양심을 부여받았으며 서로 형제애의 정신으로 행동하여야한다.

GB 18030

Chinesisch: GB 18030

> 25000 Schriftzeichen



中国标准技术开发公司　北京中易电子公司　点阵字库样本

Der universelle Zeichensatz Unicode

Der größte Zeichensatz der Welt: Unicode

Übersicht – der Zeichensatz

Zeichensatzkodierungen für Unicode

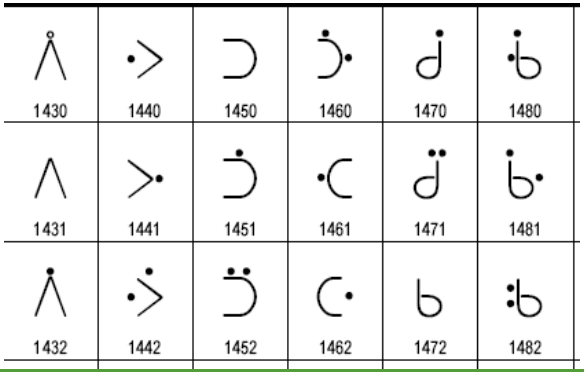
- UTF-8
- UTF-16
- UCS-2, UTF-32 (= UCS-4)

Der größte aller Zeichensätze

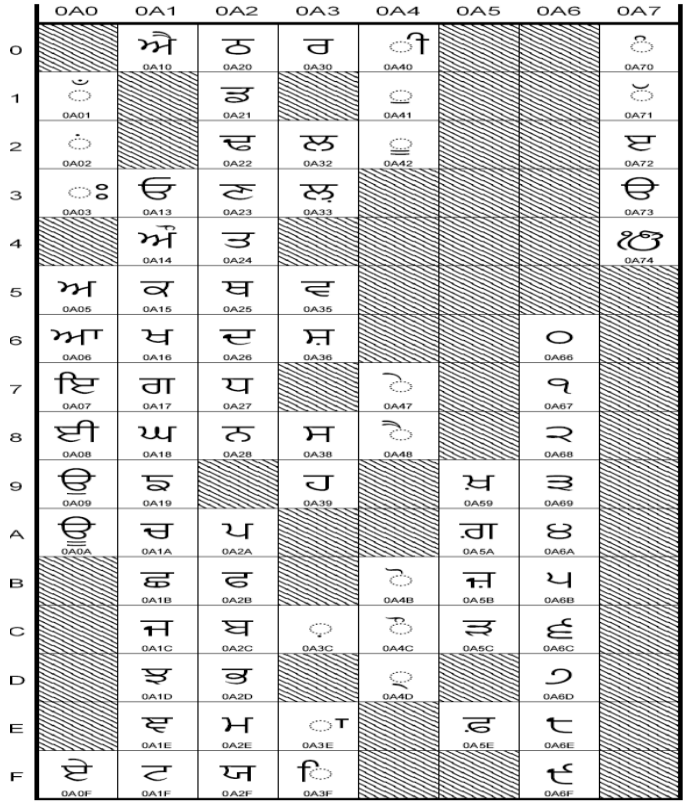
Unicode – Universeller Zeichensatz

- Die Zeichen aller Sprachen, auch toter Sprachen
- größter existierender Zeichensatz
- Version 6.2. : 110 182 Schriftzeichen (>> 65535)

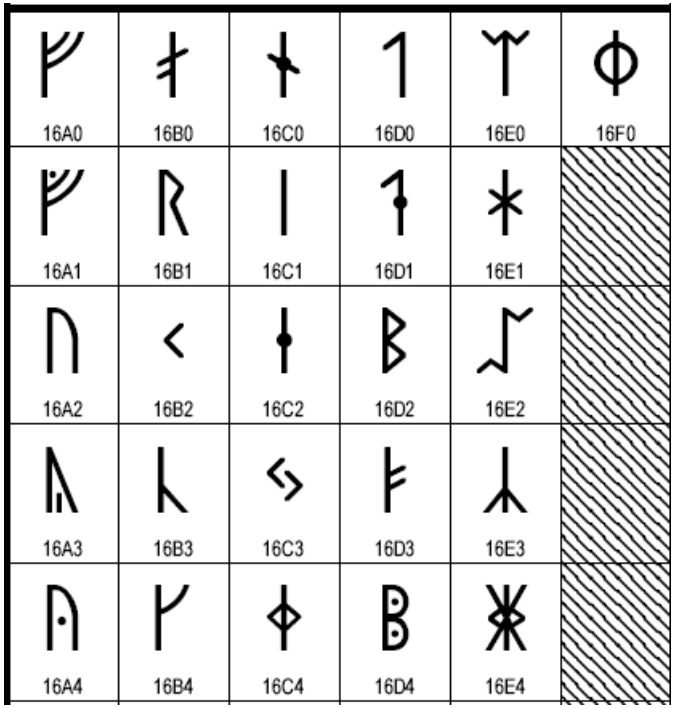
Unified Canadian Aboriginal Syllabic



Gurmukhi

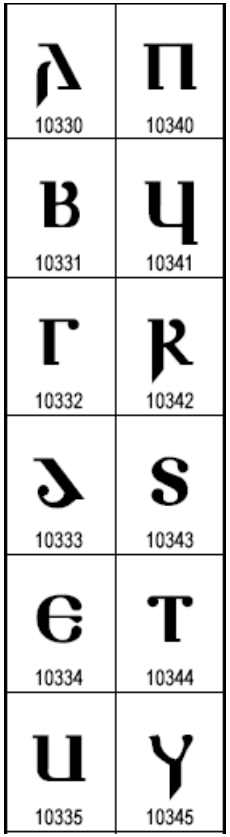


… Unicode: Historische Zeichensätze

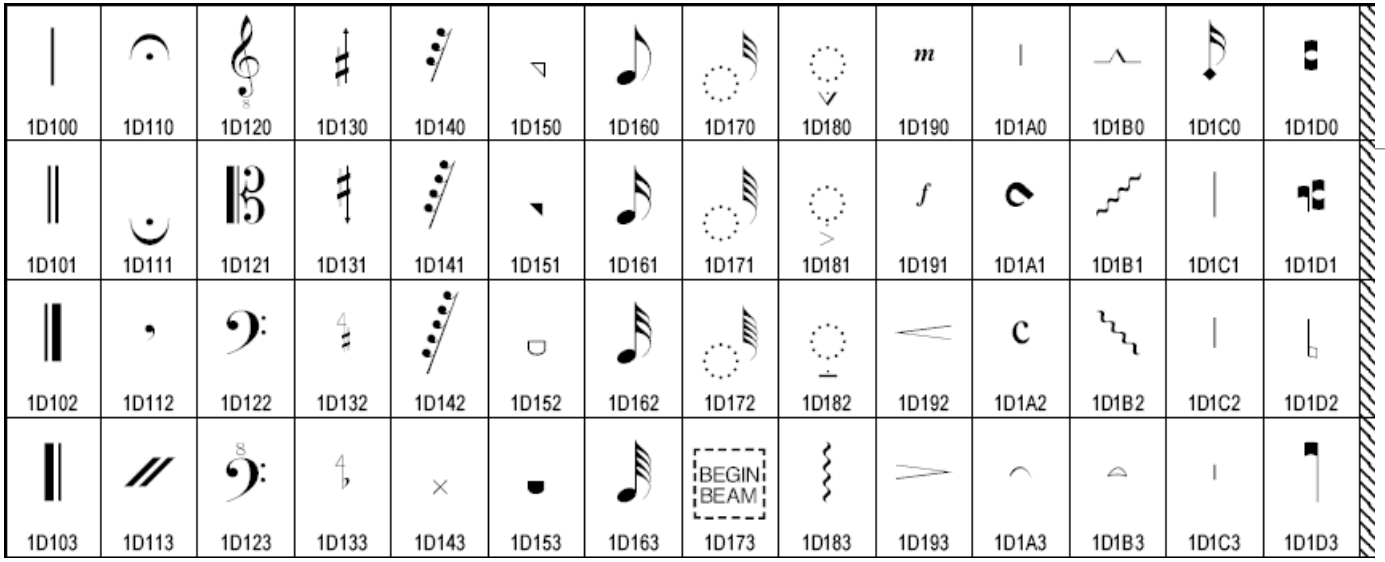


Runen

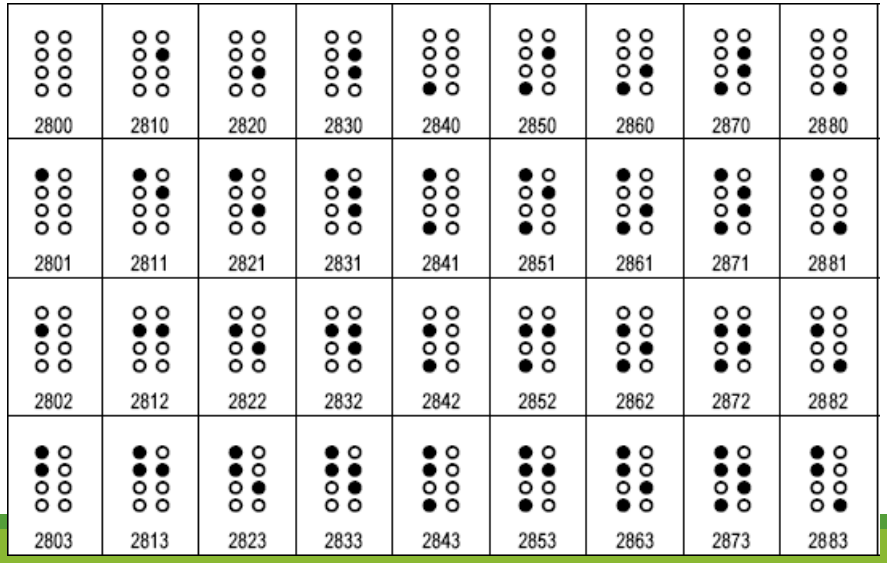
Gotisch



Musik



Braille (Blindenschrift)



Was Unicode nicht enthält...

- Klingon und einige andere künstliche Sprachen mit künstlichem Zeichensatz.

Unicode-Zeichen:Eigenschaften

Unicode-Zeichen haben Eigenschaften wie (Auswahl)

- Typ des Zeichens (z.B. Buchstabe oder Zahlwort oder Punktuation)
- Groß- oder Kleinschreibung

Außerdem haben viele Zeichen Beziehungen zu anderen Zeichen:

- Großbuchstabe ←→ Kleinbuchstabe

Diese Eigenschaften sind in der UNICODE CHARACTER DATABASE (UCD) festgehalten. Die Eigenschaftstabellen sind als Textdateien kodiert.

Die Datei die die wichtigsten Eigenschaften aller Zeichen kodiert ist

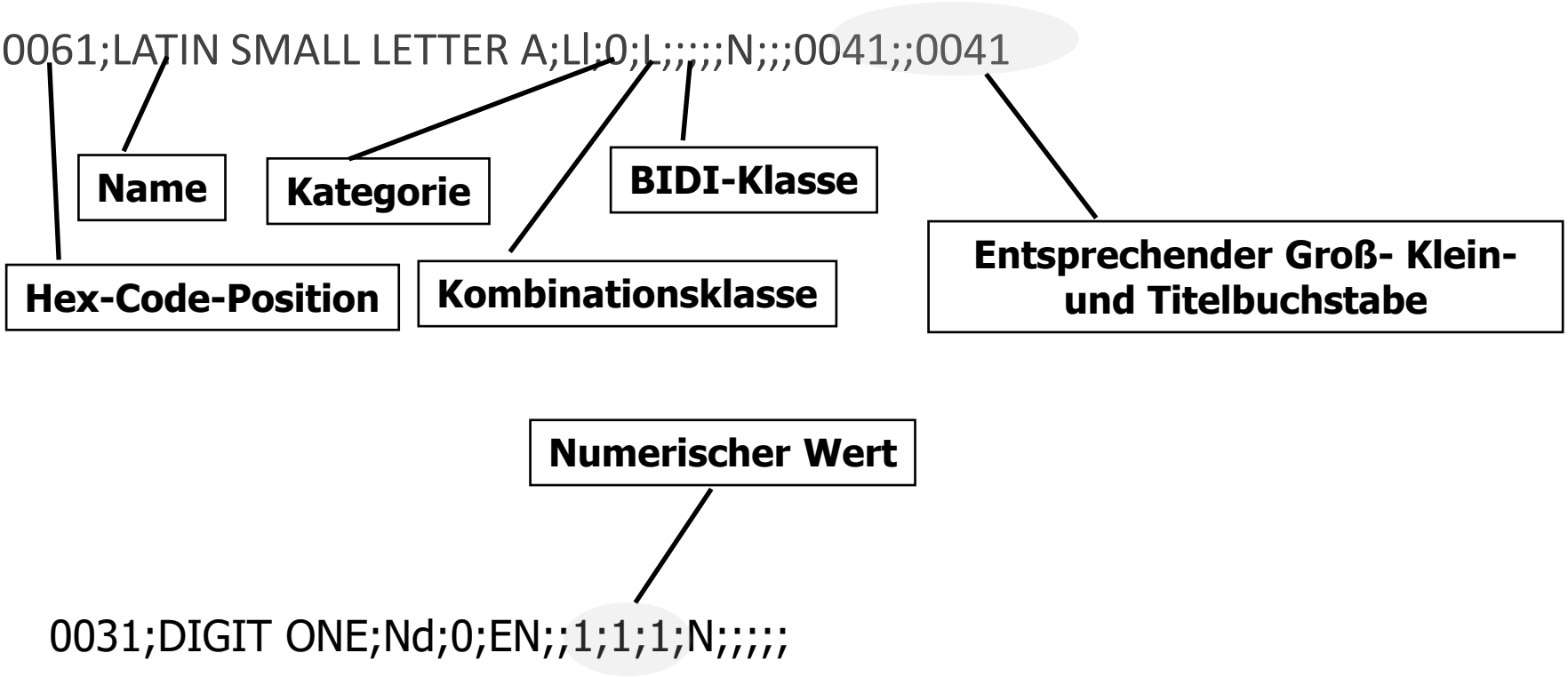
- http://www.unicode.org/Public/UNIDATA/UnicodeData.txt

UnicodeData.txt			
Name* (<reserved>)	M	N	(1) These names match exactly the names published in the code charts of the Unicode Standard.
General_Category (Cn)	E	N	(2) This is a useful breakdown into various character types which can be used as a default categorization in implementations.
Canonical_Combining_Class (0)	N	N	(3) The classes used for the Canonical Ordering Algorithm in the Unicode Standard.
Bidi_Class (L, AL, R)	E	N	(4) These are the categories required by the Bidirectional Behavior Algorithm in the Unicode Standard.
Decomposition_Type (None) Decomposition_Mapping (=)	E	N S	(5) This field contains both values, with the type in angle brackets.
Numeric_Type (None) Numeric_Value (Not a Number)	E	N N	(6) If the character has the *decimal digit* property then the value of that digit is represented with an integer value in fields 6, 7, and 8.
	E	N	(7) If the character has the *digit* property, but is not a decimal digit, then the value of that digit is represented with an integer value in fields 7 and 8.
	E	N	(8) If the character has the *numeric* property, as specified in Chapter 4 of the Unicode Standard, the value of that character is represented with an positive or negative integer or rational number in this field.
Bidi_Mirrored (N)	B	N	...
Unicode_1_Name (<none>)	M	I	(10) This is the old name as published in Unicode 1.0.
ISO_Comment (<none>)	M	I	(11) This is the ISO 10646 comment field.
Simple_Uppercase_Mapping	S	N	(12) Simple uppercase mapping (single character result).
Simple_Lowercase_Mapping	S	N	(13) Simple lowercase mapping (single character result).
Simple_Titlecase_Mapping	S	N	(14) Similar to Uppercase mapping (single character result).

Examples from UnicodeData.txt

0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;;0061;

0061;LATIN SMALL LETTER A;Ll;0;L;;;;;N;;;0041;;0041



0031;DIGIT ONE;Nd;0;EN;;1;1;1;N;;;;;

Zeichensatz**kodierung** für Zeichensätze mit mehr als 256 Zeichen am Beispiel Unicode

Methoden:

- Variable Länge, byte-basiert – z.B. UTF-8
  - Für jedes Zeichen werden >= 1 Byte verwendet
    - in den meisten Fällen ASCII-kompatibel
    - manchmal auch nicht (SHIFT-JIS)
- Variable Länge, aber immer > 1 Byte (z. B. UTF-16)
  - Selbe Länge (2 Byte) für fast alle Zeichen; einige selten verwendete Zeichen > 2 Byte
- Feste Länge (e.g. UCS-4 / UTF-32)
  - Selbe Bytezahl für alle Zeichen

Unicode-Zeichensatzkodierungen

UTF-7: Variable Länge, 1-4 Byte, nur 7 Bit; wenig verwendet

UTF-8: Variable Länge 1-4 Byte; sehr weit verbreitet

UTF-16: Variable Länge, aber meiste Zeichen mit 2 Byte kodiert

basiert auf UCS-2; Java verwendet UTF-16 als interne Zeichensatzkodierung

UCS-2 : Fixe Länge (2 Bytes max. 65535 Zeichen); VERALTET

UCS-4 (≈ UTF-32) : Fixe Länge (4 Bytes), braucht viel Speicherplatz...

Mehrbyte-Zeichensatzkodierungen mit variabler Länge

Grundlagen zum Verständnis von UTF-8

Was heißt ASCII-kompatibel?

Eine Zeichensatzkodierung ist ASCII-kompatibel, wenn jedes Byte >= 0 & <= 127 den entsprechenden ASCII-Buchstaben repräsentiert. Alle Zeichen, die keine ASCII-Zeichen sind, werden durch Bytesequenzen repräsentiert, in denen jedes Byte einen Wert >= 128 (<=255)

→das hat folgende Konsequenzen für Programme, die nur ASCII verarbeiten können:

- grundlegende Texttokenisierung kann vorgenommen werden

- z.B. funktioniert eine HTML-Parser

  - alle Funktionalitäten, die Bytes > 128 nicht speziell implementieren, sind verfügbar.

UTF-8 : Wie funktionierts?

UTF-8 – Wie es funktioniert

Unicode – kurze Wiederholung

Jedes Unicode Zeichen hat einen numerischen Kode

Es gibt >> 65000 Unicode Zeichen (aber immer < 1 Mio)

- die numerischen Kodes < 128 sind ASCII-äquivalent
- die numerischen Kodes < 256 sind == Iso-8859-1

UTF-8: Anforderungen

Anforderungen an UTF-8

- Kodiere mehrere 100 000 Zahlen als Sequenzen von 1-4 Bytes
- Ascii-kompatibel
- Schnelles Auffinden des nächsten Zeichens an beliebiger Stelle einer Bytefolge.
- Einfach zu kodieren-dekodieren

UTF-8 Spezifikation 1

Für jeden Kode c produziere eine Bytesequenz folgender Form:

if c < 128 (2^7)

CS = c

if c >= 128 and < 2048 (2^{11})

Das erste Byte ist 110YYYYY, folgendes Byte ist 10XXXXXX, c wird in den Y und X-Positionen kodiert, wobei die ersten >=2 Bytes in die YYYYY Sequenz verschoben werden.

INTEGER 00000000 00000000 00000000 11111111 (= 255) wird

11000011 10111111

Wann ist Schluss für zwei Bytes? bei >= 2048 (2^{11})

UTF-8 Spezifikation 2

if c >= 2048 (2^{11}) and c <= 65535 (2^{16})

Erstes Byte ist IIII0YYYY, die folgenden zwei Bytes sind I0XXXXXX, c wird wieder in YYYY und XXXXXX kodiert; wobei die führenden Bytes verschoben werden.

Die Integer 00000000 00000000 11111111 11111111 (= 65535)

wird

11101111 10111111 10111111

UTF-8 Spezifikation 3

if c >= 65536 (2^{16}) (und, was immer der Fall ist < 2097152 (2^{21}))

erstes Byte ist IIII0YYY, die drei folgenden I0XXXXXX

UTF-8 nach Unicode: in Perl

```
sub utf8_to_unicode
{
my $sequence = shift;
my @byte_sequence = split //, $sequence;
my $i  = 0;
$val1 = unpack("C",$byte_sequence[$i]);
if($val1 < 128)
{
   return $val1;
}
elsif($val1 >= 128 && $val1 < 192) #Bits: I0??????
{
  die "Invalid leadings byte in utf-8 value: $_";
}
elsif($val1 >= 192 && $val1 <224) #Bits: II0?????
{
  $i++;
  $val2 = unpack("C",$byte_sequence[$i]);
  if($val2 < 128 || $val2 >= 192)
  {
    die "Invalid 2-byte utf-8 value in $file , $line: $_";
  }
  my $unicode_value = (($val1^192)<<6)| ($val2^128);
  return $unicode_value;
}
elsif($val1 >= 224 && $val1 <240)  #Bits: III0????
{
      $i++;
      $val2 = unpack("C",$byte_sequence[$i]);
      $i++;
      $val3 = unpack("C",$byte_sequence[$i]);
      if($val2 < 128 || $val2 >= 192
        ||$val3 < 128 || $val3 >= 192)
      {
       die "Invalid 3-byte utf-8 value in $file , $line: $_";
      }
      my $unicode_value = (($val1^224)<<12)|
      (($val2^128)<<6)|($val3^128);
      return $unicode_value;
}
#4-Byte Sequenzen entsprechen; hier nicht behandelt
else
{
      die "Unicode values in this range not supported  $_";
}
```

Unicode-Kodepunkt-nach UTF-8 (C++)

```
int convert_unicode_to_utf8(unsigned char *output_string,int max,int input_char)
{
 if(input_char < 128) {
  *output_string = input_char;
  output_string++;
  return 1;
 }
 else if(input_char < 2048 && max >1) {
  //ending 6 bits will be written into second byte; set first two leading bits to 1
  *output_string = (unsigned char)192|input_char>>6;
  output_string++;
   //only take six last bits; set leading bit to 1; second bit to 0
   //~64 is l0IIIIII; bitwise 'and' with this clears second bit
  *output_string = ((unsigned char)128|input_char)&(~64);
  output_string++;
  return 2; //number of bytes written
 }
 else if(input_char < 65536 && max >2) {
  //set first 3 leading bits to 1
  *output_string = (unsigned char)224|input_char>>12;
  output_string++;
  //only take six last bits; set leading bit to 1; second bit to 0
  *output_string = ((unsigned char)128|input_char>>6)&(~64);
  output_string++;
  //only take six last bits; set leading bit to 1; second bit to 0
  *output_string = ((unsigned char)128|input_char)&(~64);
  output_string++;
  return 3; //number of bytes written

 }
 else if(input_char < 2097152 && max > 3) {
  //set first 4 leading bits to 1
  *output_string = (unsigned char)240|input_char>>18;
  output_string++;
  //only take six last bits; set leading bit to 1; second bit to 0   *output_string = ((unsigned
       char)128|input_char>>12)&(~64);
  output_string++;
  //only take six last bits; set leading bit to 1; second bit to 0
  *output_string = ((unsigned char)128|input_char>>6)&(~64);
  output_string++;
  //only take six last bits; set leading bit to 1; second bit to 0
  *output_string = ((unsigned char)128|input_char)&(~64);
  output_string++;
  return 4; //number of bytes written
 }
 else
  return 0; //certainly not a valid unicode character
}
```

Weitere Unicode-Kodierungen

Weitere Unicode-Kodierungen

UCS-2, UTF-32, UTF-16

UCS-2, UTF-32 (UCS-4)

UCS-2 (veraltet), UTF-32 (≈ UCS-4)

Diese Zeichensatzkodierungen kodieren Unicode als Bytesequenzen fester Länge:

UCS-2 auf 2 Byte, max 2^{16} Zeichen, deshalb veraltet

- ersetzt durch UTF-16, s.u.

UTF-32 auf 4 Byte

Weitere Zeichensatzkodierungen für Unicode

UTF-16 – variable Bytelänge (2 oder 4 Byte)

UTF-16 (Erweiterung für UCS-2): kodiert nur Zeichen> 65535 as 4-Byte-Sequenzen; für alle anderen Zeichen, UCS-2-compatible, 2 Bytes pro Zeichen).

UTF-16 repräsentiert Buchstaben über UxFFFF (65535) als Surrogatpaare (surrogate pairs) aus dem Bereiche xD800-xDFFF (die in Unicode keinem Zeichen zugewiesen sind)

Endianess

Kodierungsproblem bei UTF-16, UTF-32 und UCS-4

Unterschiedliche Rechnerarchitekturen kodieren mehr-Byte-Zahlen unterschiedlich:

- **Little Endian**: Byte mit hohem Wert zuerst; Zahl eins auf zwei Byte kodiert als:
  00000001 00000000

- **Big Endian**: Byte mit niedrigem Wert zuerst; Zahl eins:

  00000000 00000001

- Daher wird bei USC-32, UCS-2 und UTF-16 (s.u.) am Anfang der Datei der Buchstabe FEFF (nicht sichtbarer Leerschritt) eingesetzt (Byte Order Mark oder BOM). Aus dem Umstand, ob er als FE-FF oder FF-FE erscheint, lässt sich die Endianess (Bytereihenfolge) erkennen.

- Alternative: Angabe der Zeichensatzkodierung als LE/BE, z.B. UTF-32LE, UTF-32BE

- dies ist nur beim Einlesen/Auslesen/Übermitteln von Binärdaten zwischen Rechnern relevant

Andere Mehrbyte-Kodierungen

Andere Mehrbytekodierungen

Neben Unicode gibts es noch andere Zeichensätze, die mehr als 256 Zeichen enthalten, und die daher nicht in einem Byte kodiert werden können. Die bekanntesten Zeichensätze dieser Art sind die Zeichensätze für Chinesisch, Japanisch und Koreanisch (engl. auch CJK).

ASCII-kompatible Zeichensatzkodierungen

Dies meisten CJK-Zeichensatzkodierungen mit variabler Länge sind ASCII-kompatibel

- GB-1280, GBK, GB 18300 (Chinesisch)

- EUC-JP, EUC-KR (Japanisch/Koreanisch)

(funktionieren nach ähnlichen Prinzipien wie UTF-8 für Unicode)

Nicht Ascii-kompatible Mehrbyte-Zeichensatzkodierungen

Es gibt einige Multibyte-Zeichensatzkodierungen, die nicht ASCII-kompatibel sind.

a) Shift-Jis (Japanisch. Nur erstes Byte einer Folge muss >= 128 sein)

日本語、半角、Shift-JIS

インターネットで受け取ったメールが文字化けしていて、「半角カナを使っているから」といわれたことはありませんか？　あるいは送信コードを「Shift JISではなくJISにしてください」と注意された経験は？

b) Zeichensatzkodierungen, die ausschließlich im ASCII-Bereich arbeiten (7-Bit) und mit Escape-Sequenzen arbeiten, um zwischen ASCII und anderen Zeichensätzen zu wechseln)

- ISO-2022-(JP/KR/CN)

ISO-2022-JP (Escape-Sequenzen in Rot)

```
<LI>$BI,MW$J%G!<%?$,>C5n$5$l$
k(J

<LI>$BF|IU@_Dj$,$G$-
$J$/$J$k(J
```

Andere Eigenschaften

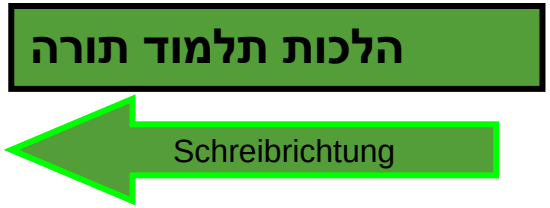
Zusätzliche Eigenschaften von Zeichensatzkodierungen:

- Behandlung der Schriftrichtung (v.a. Hebräisch)

Hebräische Schriftrichtung

Visuelle vs. logisches Hebräisch (visual/logical)

- Hebräisch wird von rechts nach links geschrieben
- Die sogenannte visuelle Zeichensatzkodierung für das Hebräische (iso-8859-6) verwendet die falsche Byte-Reihenfolge – der erste Buchstabe der Bytesequenz, die eine Zeile repräsentiert ist das letzte Byte des Textes
- die kommt daher, das Browser früher den Text nicht von rechts nach links darstellen konnten; daher drehte man den Text einfach um
- funktioniert nur zeilenweise!!!
- zur Weiterverarbeitung und Konversion nach Unicode muss die Zeile umgedreht werden
- Für Arabisch (ebenfall rechts-links) gibt es keine visuelle Kodierung, da es auf frühen Browsern sowieso nicht dargestellt werden konnte.



Kodierungskonversion

Werkzeuge zur Kodierungskonversion:

- iconv (Programmbibliothek & ausführbares Programm)
  Bsp: iconv –f iso-8859-1 –t utf-8 < eingabedatei > ausgabedatei

- recode (ausführbares Programm)

- Word: Abspeichern von Dateien als Text in beliebigem Zeichensatz (zur gelegentlichen Konversion; nicht zur Automatisierung geeignet)

- Eigenes Programm auf Basis einer Skriptsprache (s. nächste Folie)

Kodierungskonversion II

Kodierungskonversion in Programmiersprachen:

Unterstützung von Zeichensatzkodierung:

Beim Lesen und Abspeichern von Dateien

Konversion von Zeichenketten

Perl: Seit Perl 5.8. : Perl Encode

Python: codecs module

ICU-library http://site.icu-project.org/ für C++ und Java