Zeichencodierungen

Theorie

Version vom 12. Februar 2025

1 Der ASCII-Code

Ein *Code* ordnet jedem Zeichen einer Zeichenmenge eindeutig ein Zeichen oder eine Zeichenfolge einer (möglicherweise anderen) Zeichenmenge zu.

Eine *Zeichencodierung* ist ein spezieller Code, der Schriftzeichen (Buchstaben, Ziffern, Symbolen) jeweils eine Zahl zuordnet, um sie zu speichern oder zu übertragen.

Einer der ersten standardisierten Zeichencodes für Computer ist der *American Standard Code for Information Interchange* (ASCII). Die aktuell gültige Version stammt aus dem Jahr 1968 und ordnet den Gross- und Kleinbuchstaben des lateinischen Alphabets, den zehn Ziffern, einigen Interpunktions- und Sonderzeichen sowie bestimmten Steuerzeichen eindeutig eine Nummer zwischen 0 und 127 zu.

Die ASCII-Tabelle

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Bei den blau dargestellten Elementen handelt es sich um *Steuerzeichen*, die aus der Zeit der Fernschreiber stammen und heute nicht mehr oder für andere Zwecke verwendet werden.

Neben dem ASCII-Code gibt es noch einige andere Codes, die von speziellen Computersystemen gebraucht werden. Beispielsweise den *Extended Binary Coded Decimal Interchange Code* (EBCDIC) für Grossrechenanlagen der Firma IBM.

2 Die ISO-8859-X-Standards

Zu Beginn der 1980er Jahre kam der *Personal Computer* (PC) auf den Markt. Im Gegensatz zu den bis dahin vorherrschenden Grossrechenanlagen war er auch für Privatpersonen und kleinere Unternehmen erschwinglich.

Mit der weltweiten Verbreitung des PCs mussten neben den (Sonder-)Zeichen westeuropäischer Sprachen auch Arabisch, Griechisch, Hebräisch, Kyrillisch, Türkisch oder Thailändisch codiert werden. Diesen Mangel versuchte man u. a. mit den Standards ISO 8859-1 bis ISO 8859-11 sowie ISO 8859-13 bis ISO 8859-16 zu beheben, die Zeichen jeweils mit 8 Bits codieren. Diese Zeichencodierungen sind jedoch nur noch selten in Gebrauch und werden langfristig von Unicode abgelöst.

3 Unicode

3.1 Gründe für die Entstehung von Unicode

Ab 1990 wurde die Situation komplizierter, da häufiger Dokumente zwischen verschiedenen Sprachregionen ausgetauscht wurden. Warum?

1991 wurde die Version 1.0.0 des Unicode-Standards mit 7 161 Zeichen veröffentlicht, um eine Zeichencodierung einzuführen, die einen reibungslosen Datenaustausch über Sprachgrenzen hinweg ermöglichen soll.

Im Jahr 2011 umfasste der Standard bereits mehr als 100 000 Zeichen. Mittlerweile (Stand 2022) sind es 159 Schriftsysteme mit mehr als 144 000 Zeichen.

3.2 Was definiert der Unicode-Standard?

Längerfristig soll für jedes sinntragende Zeichen bzw. Textelement aller bekannten Schriftkulturen und Zeichensysteme ein eindeutiger digitaler Code festgelegt werden.

- Jedes Zeichen, das in den Standard aufgenommen wird, erhält eine *eindeutige* Nummer (*Codepoint*). Diese Nummern werden in Ebenen (*planes*) und Blöcke (*blocks*) gruppiert.

  Die Nummern werden üblicherweise hexadezimal mit mindestens vier Stellen und führenden Nullen geschrieben, denen man `U+` voranstellt.

  Beispiel: Der Umlaut „Ü“ hat den Codepoint `U+00DC`.

- Es wird festgelegt, wie die Nummern der Zeichen digital dargestellt werden. Hier definiert der Standard mehrere Varianten. Dazu gleich mehr.

Ein Zeichen mit einer Nummer darf nicht mehr umnummeriert werden. Warum?

3.3 Das Unicode Transformation Format (UTF)

Da der Unicode-Standard bereits knapp 150 000 Zeichen umfasst (Stand 2022), genügt ein Byte nicht mehr aus, um jedes Zeichen darzustellen.

1 Byte: $2^8 = 256$ Zeichen
2 Byte: $2^{16} = 65\,536$ Zeichen
3 Byte: $2^{24} = 16\,777\,216$ Zeichen
4 Byte: $2^{32} = 4\,294\,967\,296$ Zeichen

3.3.1 UTF-32

Für die binäre Darstellung eines Unicode-Zeichens werden 32 Bit (4 Byte) verwendet. Damit lassen sich problemlos alle Unicode-Zeichen darstellen.

Beispiel: lateinischer Buchstabe `'A'`

Nummer des Zeichens:

UTF-32:

Nachteil:

Vorteil:

3.3.2 UTF-16

Je nach Codenummer werden 16 oder 32 Bit verwendet. Es ist das älteste Unicode Transformationsformat und wird beispielsweise vom Betriebssystem Windows oder von der Programmiersprache Java verwendet.

Sehr viele aktuell verwendete Zeichen haben einen Codepunkt in der sogenannten *Basic Multilingual Plane* (BMP). Das ist die unterste Codeebene, die aus 256 Blöcken mit jeweils 256 Codepunkten besteht. In diesem Fall hat das Zeichen eine Nummer zwischen `U+0000` und `U+FFFF` und wird direkt durch die entsprechende 16 Bit grosse Binärzahl codiert.

Zeichen, die in einer der höheren drei Codeebenen liegen (`U+10000` bis `U+10FFFF`), werden mittels eines Tricks durch eine 32 Bit grosse Binärzahl codiert:

- Von dieser Zeichennummer wird die Zahl `0x10000` subtrahiert. Dies ergibt eine Binärzahl zwischen `0x00000` und `0xFFFFF` (20 Bits).
- Danach werden die ersten 10 Bits hinter das Präfix `110110` und die zweiten 10 Bits hinter das Präfix `110111` gesetzt.

Da Unicode-Nummern `110110XXXXXXXXXX` und `110111XXXXXXXXXX` für diesen Zweck reserviert sind, gibt es keine Verwechslungen mit den Zeichen, die eine 16-Bit-Codierung haben.

Beispiel 1

Umlaut `'Ä'`

Nummer des Zeichens: `U+00C1 = 1100|0001`

UTF-16: `00000000 11000001`

Beispiel 2

die ägyptische Hieroglyphe 

Nummer des Zeichens: `U+1304F` (grösser als `U+10000`)

Subtrahiere `0x10000`:

```
    0x1304F
−   0x10000
=   0x0304F
```

Binär: `0000|0011|0000|0100|1111`

UTF-16: `11011000 00001100 11011100 01001111`

3.3.3 UTF-8

Aufbau

Je nach Zeichennummer werden 8, 16, 24 oder 32 Bit pro Zeichen verwendet. Damit ein Code mit einer variablen Codewortlänge wieder decodiert werden kann, müssen neben der zu codierenden Information *Steuerzeichen* eingefügt werden. Hat ein Byte die Form ...

- `0XXXXXXX`, so ist es ein ASCII-Zeichen,
- `110XXXXX`, so ist es das Startbyte eines Zeichens, das aus 2 Byte besteht,
- `1110XXXX`, so ist es das Startbyte eines Zeichens, das aus 3 Byte besteht,
- `11110XXX`, so ist es das Startbyte eines Zeichens, das aus 4 Byte besteht,
- `10XXXXXX`, so ist es ein Folgebyte, das einen Teil der Zeicheninformation enhält.

Ein `X` steht symbolisch für ein beliebiges Bit (0 oder 1).

Abbildungsbereiche

1 Byte (`0XXXXXXX`):

⇒

2 Byte (`110XXXXX 10XXXXXX`):

⇒

3 Byte (`1110XXXX 10XXXXXX 10XXXXXX`):

⇒

4 Byte (`11110XXX 10XXXXXX 10XXXXXX 10XXXXXX`):

⇒

Vorteile von UTF-8:

- Kompatibilität mit den älteren Codierungsschemata ASCII und ISO-8859-X.
- Im Mittel weniger Speicherverbrauch durch variable Zeichenlänge.
- Im Gegensatz zu UTF-16 und UTF-32 lässt sich die Reihenfolge der Bytes unabhängig von der codierenden Hardware bestimmen (siehe Byte Order Mark).

Nachteile von UTF-8:

- Codieren und decodieren ist aufwändig.
- Sprachen, deren Zeichen grosse Unicode-Nummern haben, werden durch grösseren Speicherbedarf benachteiligt.

3.4 Die Byte Order Mark

Moderne Computerprozessoren fassen mehrere Bytes zu einem *Datenwort* zusammen.

Anzahl Bytes	Bezeichnung
2	WORD
4	DWORD (double word)
8	LWORD (long word)

Aus Effizienzgründen verarbeiten nicht alle Prozessortypen die Bytes eines Datenworts in der gleichen Reihenfolge.

Big-Endian-Systeme verarbeiten das höchstwertige Byte zuerst. *Beispiel:*

`A3 39 F3 74` → `A3 39 F3 74`

Little-Endian-Systeme verarbeiten das niederwertigste Byte zuerst. *Beispiel:*

`A3 39 F3 74` → `74 F3 39 A3`

Wenn Textdaten im UTF-16 oder UTF-32-Format zwischen Computersystemen ausgetauscht werden, ist die Reihenfolge der Bytes von Bedeutung. Warum ist dies bei UTF-8 kein Problem?

Wie kann ein Computer erkennen, ob er bei einem (Unicode-)Text die Bytes in einem Datenwort in umgekehrter Reihenfolge interpretieren muss?

Das Quellsystem speichert am Anfange des Textes die Byte Order Mark (BOM) ab. Im Falle der UTF-16-Codierung handelt es sich um das Unicode-Zeichen `U+FEFF`.

Wenn ein Computer vom gleichen Verarbeitungstyp dieses Zeichen liest, erkennt er, dass die folgenden Bytes in der „seiner“ Reihenfolge vorliegen. Andernfalls stösst er auf das „illegale“ Unicode-Zeichen `U+FFFE` und weiss, dass er in jedem Datenwort die Bytes von rechts nach links lesen muss.

3.5 Unicode in Python

Die Programmiersprache Python verwendet Unicode und als Standardcodierung UTF-8.

Die Funktion `ord(...)`

Die Funktion `ord('Zeichen')` gibt den dezimalen Wert des Codepoints von *Zeichen*.

```
print(ord('Ü')) # => 220 (dezimal)
```

Falls nötig, kann der dezimale Wert mit `hex(...)` hexadezimal dargestellt werden.

```
print(hex(ord('Ü'))) # => 0xdc (hexadezimal)
```

Die Funktion `chr(...)`

Die Funktion `chr(Nummer)` gibt das Unicode-Zeichen zur *Nummer* (im Zehnersystem) zurück.

```
print(chr(42)) # => '*'
```

Da die Nummern der Unicode-Zeichen oft hexadezimal geschrieben werden, muss man sie zuvor mit `int('...', base=16)` umrechnen, wenn man sie anzeigen will.

Um welche Symbole handelt es sich?

U+265E:
```
print(chr(int('265E', base=16)))
```

U+1F600:
```
print(chr(int('1F600', base=16)))
```

Eingabe von Zeichen durch Escape-Sequenzen

Eingabe mit 16 bit:
```
print('\u0043') # => 'C'
```

Eingabe mit 32 bit:
```
print('\U00000043') # => 'C'
```