Codierung von Zeichen

Dezimal	Binär		Dezimal	Binär		Dezimal	Binär	
32	00100000	SP	64	01000000	@	96	01100000	`
33	00100001	!	65	01000001	A	97	01100001	a
34	00100010	"	66	01000010	B	98	01100010	b
35	00100011	#	67	01000011	C	99	01100011	c
36	00100100	$	68	01000100	D	100	01100100	d
37	00100101	%	69	01000101	E	101	01100101	e
38	00100110	&	70	01000110	F	102	01100110	f
39	00100111	'	71	01000111	G	103	01100111	g
40	00101000	(	72	01001000	H	104	01101000	h
41	00101001	)	73	01001001	I	105	01101001	i
42	00101010	*	74	01001010	J	106	01101010	j
43	00101011	+	75	01001011	K	107	01101011	k
44	00101100	,	76	01001100	L	108	01101100	l
45	00101101	-	77	01001101	M	109	01101101	m
46	00101110	.	78	01001110	N	110	01101110	n
47	00101111	/	79	01001111	O	111	01101111	o
48	00110000	0	80	01010000	P	112	01110000	p
49	00110001	1	81	01010001	Q	113	01110001	q
50	00110010	2	82	01010010	R	114	01110010	r
51	00110011	3	83	01010011	S	115	01110011	s
52	00110100	4	84	01010100	T	116	01110100	t
53	00110101	5	85	01010101	U	117	01110101	u
54	00110110	6	86	01010110	V	118	01110110	v
55	00110111	7	87	01010111	W	119	01110111	w
56	00111000	8	88	01011000	X	120	01111000	x
57	00111001	9	89	01011001	Y	121	01111001	y
58	00111010	:	90	01011010	Z	122	01111010	z
59	00111011	;	91	01011011	[	123	01111011	{
60	00111100	<	92	01011100	\	124	01111100	|
61	00111101	=	93	01011101	]	125	01111101	}
62	00111110	>	94	01011110	^	126	01111110	~
63	00111111	?	95	01011111	_	127	01111111	DEL

Image by Gerd Altmann from Pixabay

Codierung von Zeichen - Unicode

Unicode Transformation Formate (UTF) – Implementieren den Unicode Zeichensatz. **UTF-8**, UTF-16, UTF-32

Derzeit: 1.111.998 elementare Zeichen („Codepunkte“) verfügbar
Darstellung: U+00DF (Mindestens 4x4Bit, bis zu U+10FFFF)

Unicode ist ein internationaler Standard, in dem langfristig für jedes sinnvolle Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen. Unicode wird ständig um Zeichen weiterer Schriftsysteme durch das Unicode-Konsortium ergänzt.

(Wikipedia, https://de.wikipedia.org/wiki/Unicode)

Damit geht auch: **https://unicode.org/emoji/charts/full-emoji-list.html**

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600	😀	😀	😀	😀	😀	😀	😀	😀	😀	—	—	—	grinning face

Codierung von Zeichen - Unicode

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600	😀	😀	😀	😀	😀	😀	😀	😀	😀	—	—	—	grinning face

Codepunkt („**Zeichennummer**“): **`0x1F600`**

Codierung?

Wie wird daraus ein Bitmuster, das gespeichert und verarbeitet werden kann?

Zechnall (September 2021)
Schiebel (Oktober 2023)

Codierung von Zeichen – UTF Varianten

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600										—	—	—	grinning face

Codepunkt („**Zeichennummer**“): **`0x1F600`**

UTF-32	UTF-16	UTF-8
Jeder Codepoint wird durch **4 Bytes** dargestellt. Das reicht für alle Codepoints verschwendet aber Ressourcen - einfach aber verschwenderisch.	Jeder Codepoint wird durch **2 Bytes** dargestellt, halbiert also den Speicherverbrauch von UTF-32. 2 Bytes reichen nicht für alle Codepoints, deswegen muss man tricksen, indem man die oben erwähnte Lücke zwischen U+D7FF und U+E000 nutzt.	UTF-8 nutzt 8 Bit lange Codeblöcke. UTF-8 verwendet je nach Bedarf zwischen 1 und 4 Byte, um einen Codepoint darzustellen. **Ein codiertes Zeichen hat also keine feste Bitlänge**.

Codierung von Zeichen - UTF-8

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600	😀	😀	😀	😀	😀	😀	😀	😀	😀	—	—	—	grinning face

Warum UTF-8?

- Die ersten 127 Zeichen und Bytes sind identisch mit ASCII, d. h. alle Texte, die in der Hauptsache Unicode-Zeichen mit Codepoints zwischen U+0000 und U+007F verwenden, bleiben problemlos lesbar.

- Auch Systeme, die UTF-8 nicht verstehen, können die Bytes trotzdem (eingeschränkt) verarbeiten, weil es sich um – wenn auch eher abstruse – „normale“ Zeichen handelt.

- Selbst das byteorientierte Sortieren von UTF-8-Texten funktioniert und sortiert die Strings geordnet nach dem Zahlenwert der enthaltenen Codepoints.

- Bei Verwendung von westeuropäischen Sprachen wird im Vergleich zu UTF-16 viel Speicherplatz gespart, da die meisten Zeichen nur ein Byte benötigen.

.

Zechnall (September 2021)
Schiebel (Oktober 2023)

UTF-8 Codierung mit bis zu 4 Bytes

Regeln:

- Ist das erste Bit eine 0, wird das Zeichen mit 1 Byte codiert
  → 7 Bit zur Codierung übrig, diese entspricht ASCII
- Sonst beginnt das erste Byte mit sovielen 1en, wie zur Codierung Bytes verwendet werden, gefolgt von einer 0. Alle weiteren Bytes beginnen mit „10“ – **pro Byte** können dann also **6Bit** „Nutzdaten“ untergebracht werden.

Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym
0	00	null	32	20	Leer-zeichen	64	40	@	96	60	`	128	80		160	A0		192	C0	À	224	E0	à
1	01		33	21	!	65	41	A	97	61	a	129	81		161	A1	¡	193	C1	Á	225	E1	á
2	02		34	22	"	66	42	B	98	62	b	130	82		162	A2	¢	194	C2	Â	226	E2	â
3	03		35	23	#	67	43	C	99	63	c	131	83		163	A3	£	195	C3	Ã	227	E3	ã
4	04		36	24	$	68	44	D	100	64	d	132	84		164	A4	¤	196	C4	Ä	228	E4	ä
5	05		37	25	%	69	45	E	101	65	e	133	85		165	A5	¥	197	C5	Å	229	E5	å
6	06		38	26	&	70	46	F	102	66	f	134	86		166	A6	¦	198	C6	Æ	230	E6	æ
7	07		39	27	'	71	47	G	103	67	g	135	87		167	A7	§	199	C7	Ç	231	E7	ç
8	08	back-space	40	28	(	72	48	H	104	68	h	136	88		168	A8	¨	200	C8	È	232	E8	è
9	09	horizontal tab	41	29	)	73	49	I	105	69	i	137	89		169	A9	©	201	C9	É	233	E9	é
10	0A	line feed	42	2A	*	74	4A	J	106	6A	j	138	8A		170	AA	ª	202	CA	Ê	234	EA	ê
11	0B		43	2B	+	75	4B	K	107	6B	k	139	8B		171	AB	«	203	CB	Ë	235	EB	ë
12	0C		44	2C	,	76	4C	L	108	6C	l	140	8C		172	AC	¬	204	CC	Ì	236	EC	ì
13	0D	carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D		173	AD	-	205	CD	Í	237	ED	í
14	0E		46	2E	.	78	4E	N	110	6E	n	142	8E		174	AE	®	206	CE	Î	238	EE	î
15	0F		47	2F	/	79	4F	O	111	6F	o	143	8F		175	AF	¯	207	CF	Ï	239	EF	ï
16	10		48	30	0	80	50	P	112	70	p	144	90		176	B0	°	208	D0	Ð	240	F0	ð
17	11		49	31	1	81	51	Q	113	71	q	145	91		177	B1	±	209	D1	Ñ	241	F1	ñ
18	12		50	32	2	82	52	R	114	72	r	146	92		178	B2	²	210	D2	Ò	242	F2	ò
19	13		51	33	3	83	53	S	115	73	s	147	93		179	B3	³	211	D3	Ó	243	F3	ó
20	14		52	34	4	84	54	T	116	74	t	148	94		180	B4	´	212	D4	Ô	244	F4	ô
21	15		53	35	5	85	55	U	117	75	u	149	95		181	B5	µ	213	D5	Õ	245	F5	õ
22	16		54	36	6	86	56	V	118	76	v	150	96		182	B6	¶	214	D6	Ö	246	F6	ö
23	17		55	37	7	87	57	W	119	77	w	151	97		183	B7	·	215	D7	×	247	F7	÷
24	18		56	38	8	88	58	X	120	78	x	152	98		184	B8	¸	216	D8	Ø	248	F8	ø
25	19		57	39	9	89	59	Y	121	79	y	153	99		185	B9	¹	217	D9	Ù	249	F9	ù
26	1A		58	3A	:	90	5A	Z	122	7A	z	154	9A		186	BA	º	218	DA	Ú	250	FA	ú
27	1B		59	3B	;	91	5B	[	123	7B	{	155	9B		187	BB	»	219	DB	Û	251	FB	û
28	1C		60	3C	<	92	5C	\	124	7C	|	156	9C		188	BC	¼	220	DC	Ü	252	FC	ü
29	1D		61	3D	=	93	5D	]	125	7D	}	157	9D		189	BD	½	221	DD	Ý	253	FD	ý
30	1E		62	3E	>	94	5E	^	126	7E	~	158	9E		190	BE	¾	222	DE	Þ	254	FE	þ
31	1F		63	3F	?	95	5F		127	7F		159	9F		191	BF	¿	223	DF	ß	255	FF	ÿ

Beispiel 1:

k → U+006B → $6B_{16}$ → **0**110 1011_2 (Erstes Bit ist eine Null)
→ die letzten 7Bit werden verwendet, um zu codieren, also ein „ASCII k“ in UTF-8
→ **0110 1011**

Zechnall (September 2021)
Schiebel (Oktober 2023)

UTF-8 Codierung mit bis zu 4 Bytes

Regeln:

- Ist das erste Bit eine 0, wird das Zeichen mit 1 Byte codiert
  → 7 Bit zur Codierung übrig, diese entspricht ASCII
- Sonst beginnt das erste Byte mit sovielen 1en, wie zur Codierung Bytes verwendet werden, gefolgt von einer 0. Alle weiteren Bytes beginnen mit „10“ – **pro Byte** können dann also **6Bit** „Nutzdaten“ untergebracht werden.

Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym	Dez	Hex	Sym
0	00	null	32	20	Leer-zeichen	64	40	@	96	60	`	128	80		160	A0		192	C0	À	224	E0	à
1	01		33	21	!	65	41	A	97	61	a	129	81		161	A1	¡	193	C1	Á	225	E1	á
2	02		34	22	"	66	42	B	98	62	b	130	82		162	A2	¢	194	C2	Â	226	E2	â
3	03		35	23	#	67	43	C	99	63	c	131	83		163	A3	£	195	C3	Ã	227	E3	ã
4	04		36	24	$	68	44	D	100	64	d	132	84		164	A4	¤	196	C4	Ä	228	E4	ä
5	05		37	25	%	69	45	E	101	65	e	133	85		165	A5	¥	197	C5	Å	229	E5	å
6	06		38	26	&	70	46	F	102	66	f	134	86		166	A6	¦	198	C6	Æ	230	E6	æ
7	07		39	27	'	71	47	G	103	67	g	135	87		167	A7	§	199	C7	Ç	231	E7	ç
8	08	back-space	40	28	(	72	48	H	104	68	h	136	88		168	A8	¨	200	C8	È	232	E8	è
9	09	horizontal tab	41	29	)	73	49	I	105	69	i	137	89		169	A9	©	201	C9	É	233	E9	é
10	0A	line feed	42	2A	*	74	4A	J	106	6A	j	138	8A		170	AA	ª	202	CA	Ê	234	EA	ê
11	0B		43	2B	+	75	4B	K	107	6B	k	139	8B		171	AB	«	203	CB	Ë	235	EB	ë
12	0C		44	2C	,	76	4C	L	108	6C	l	140	8C		172	AC	¬	204	CC	Ì	236	EC	ì
13	0D	carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D		173	AD	-	205	CD	Í	237	ED	í
14	0E		46	2E	.	78	4E	N	110	6E	n	142	8E		174	AE	®	206	CE	Î	238	EE	î
15	0F		47	2F	/	79	4F	O	111	6F	o	143	8F		175	AF	¯	207	CF	Ï	239	EF	ï
16	10		48	30	0	80	50	P	112	70	p	144	90		176	B0	°	208	D0	Ð	240	F0	ð
17	11		49	31	1	81	51	Q	113	71	q	145	91		177	B1	±	209	D1	Ñ	241	F1	ñ
18	12		50	32	2	82	52	R	114	72	r	146	92		178	B2	²	210	D2	Ò	242	F2	ò
19	13		51	33	3	83	53	S	115	73	s	147	93		179	B3	³	211	D3	Ó	243	F3	ó
20	14		52	34	4	84	54	T	116	74	t	148	94		180	B4	´	212	D4	Ô	244	F4	ô
21	15		53	35	5	85	55	U	117	75	u	149	95		181	B5	µ	213	D5	Õ	245	F5	õ
22	16		54	36	6	86	56	V	118	76	v	150	96		182	B6	¶	214	D6	Ö	246	F6	ö
23	17		55	37	7	87	57	W	119	77	w	151	97		183	B7	·	215	D7	×	247	F7	÷
24	18		56	38	8	88	58	X	120	78	x	152	98		184	B8	¸	216	D8	Ø	248	F8	ø
25	19		57	39	9	89	59	Y	121	79	y	153	99		185	B9	¹	217	D9	Ù	249	F9	ù
26	1A		58	3A	:	90	5A	Z	122	7A	z	154	9A		186	BA	º	218	DA	Ú	250	FA	ú
27	1B		59	3B	;	91	5B	[	123	7B	{	155	9B		187	BB	»	219	DB	Û	251	FB	û
28	1C		60	3C	<	92	5C	\	124	7C	|	156	9C		188	BC	¼	220	DC	Ü	252	FC	ü
29	1D		61	3D	=	93	5D	]	125	7D	}	157	9D		189	BD	½	221	DD	Ý	253	FD	ý
30	1E		62	3E	>	94	5E	^	126	7E	~	158	9E		190	BE	¾	222	DE	Þ	254	FE	þ
31	1F		63	3F	?	95	5F		127	7F		159	9F		191	BF	¿	223	DF	ß	255	FF	ÿ

Beispiel 2:

ä → U+00E4 → $E4_{16}$ → **1110 0100**$_2$ → Ein Byte lang, aber „1“ am Start, also MultiByte Codierung.

→ 8 Bit Daten zu codieren, dafür braucht man 2 Byte.
→ Der UTF-8 Code beginnt also mit der Startsequenz **110**
→ Dann von hinten 6 Bit → **10 0100**, das Byte beginnt mit **10** (Regel)
→ 2. Byte: **1010 0100**
→ Bleiben **11** der Daten → Auffüllen auf 5 Bit **0 0011,** Startsequenz hat 3 Bit
→ 1. Byte: **1100 0011**
→ UTF-8: **1100 0011 1010 0100**

Zechnall (September 2021)
Schiebel (Oktober 2023)

UTF-8 Codierung mit bis zu 4 Bytes

Regeln:

- Ist das erste Bit eine 0, wird das Zeichen mit 1 Byte codiert
  → 7 Bit zur Codierung übrig, diese entspricht ASCII
- Sonst beginnt das erste Byte mit sovielen 1en, wie zur Codierung Bytes verwendet werden, gefolgt von einer 0. Alle weiteren Bytes beginnen mit „10“ – **pro Byte** können dann also **6Bit** „Nutzdaten“ untergebracht werden.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
4E00	一	丁	丂	七	丄	丅	丆	万	丈	三	上	下	丌	不	与	丐
4E10	丑	丒	专	且	丕	世	丗	丘	丙	业	丛	东	丝	丞	丟	
4E20	丠	両	丢	丣	两	严	並	丧	丨	丩	个	丫	丬	中	丮	丯
4E30	丰	丱	串	丳	临	丵	丶	丷	丸	丹	为	主	丼	丽	举	丿
4E40	乀	乁	乂	乃	乄	久	乆	乇	么	义	乊	之	乌	乍	乎	乏
4E50	乐	乑	乒	乓	乔	乕	乖	乗	乘	乙	乚	乛	乜	九	乞	也
4E60	习	乡	乢	乣	乤	乥	书	乧	乨	乩	乪	乫	乬	乭	乮	乯

Beispiel 3:

乔 → U+4E54 → $4E54_{16}$ → **0100 1110 0101 0010**$_2$

→ 16 Bit Daten zu codieren, dafür braucht man 3 Byte (3 x 6 = 18)

→ Der UTF-8 Code beginnt also mit der Startsequenz **1110**

→ Dann von hinten 6 Bit → **01 0010** , das Byte beginnt mit **10** (Regel) → **1001 0010**

→ Die nächsten 6 Bit → **1110 01** → **1011 1001**

→ Die fehlenden 4 Bit → **0100,** vorgestellet die Startsequenz → **1110 0100**

→ UTF-8 Codiert: **1110 0100 1011 1001 1001 0010**

Modellvorstellung

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600 0x1F600	😀	😀	😀	😀	😀	😀	😀	😀	😀	—	—	—	grinning face

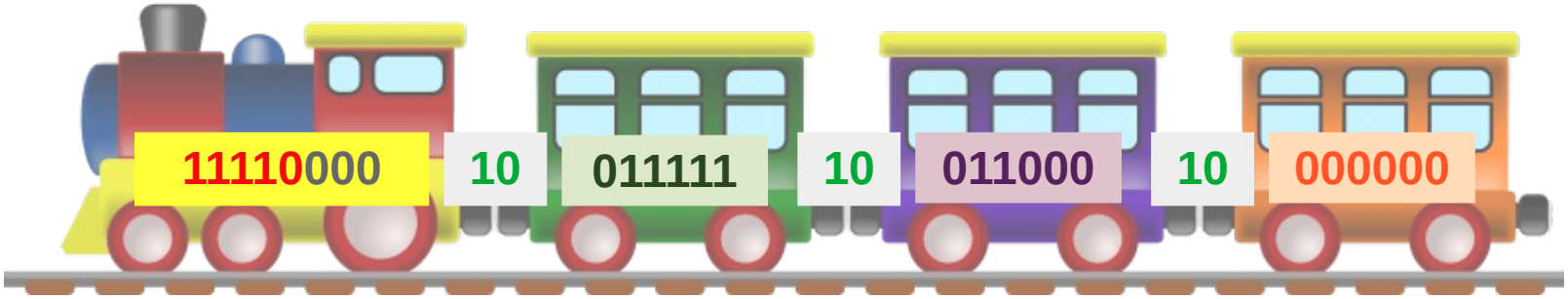
(1) Codepoint Hexadezimal → Codepoint binär

0x1F600

(2) 6-Bit Ladungen – passt noch was in die Lock?

1 Byte - 7 Bit	7 Bit in der Lok
2 Byte - 11 Bit	5 Bit in der Lok + 6 Bit
3 Byte - 16 Bit	4 Bit in der Lok + 6 Bit + 6 Bit
4 Byte - 21 Bit	3 Bit in der Lok + 6 Bit + 6 Bit + 6 Bit

(3) UTF-8 Zug, die Kupplungen sind **10**



(4) „UTF-8 Lok“, kennt die **Zahl der Zugteile + 0 als Trenner** + Padding + Ladung der Lok → Anhänger mit je 6 Bit