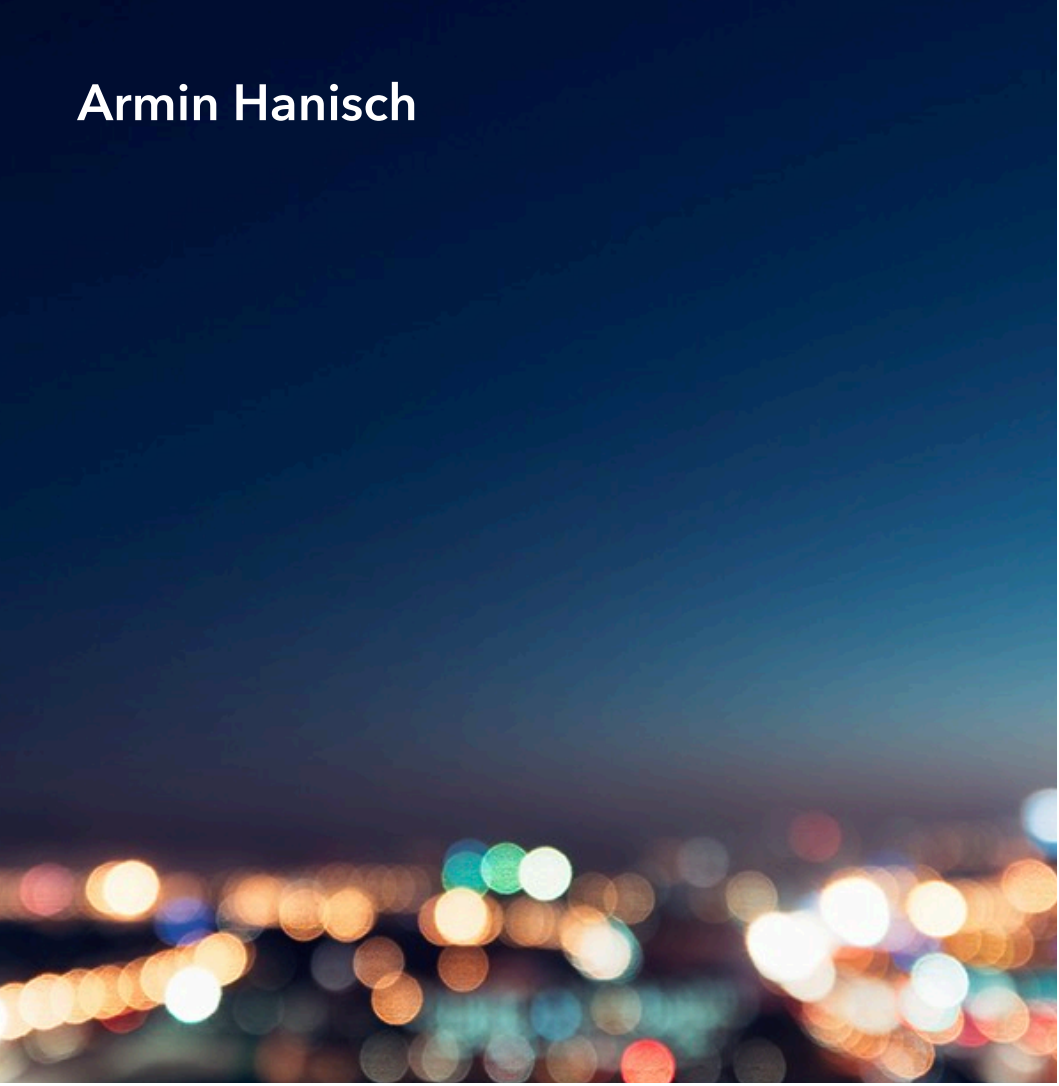
Armin Hanisch



CODES & ENCODINGS

Armin Hanisch

CODES & ENCODINGS

WIE AUS BITS ZEICHEN WERDEN UND
WAS DAS MIT EMOJI ZU TUN HAT

Dieses Werk steht unter der **Bildungslizenz (EduLicense 1.1.2)**

Das Urheberrecht für diese Zusammenstellung liegt bei Armin Hanisch. Die Weitergabe und Nutzung dieser Inhalte sowie die Veränderung ist gestattet, sofern die folgenden Bedingungen eingehalten werden:

Namensnennung
Die Weitergabe muss als Autor „Armin Hanisch (www.arminhanisch.de)" nennen. Dazu muss der Text dieser Lizenz angegeben werden. Dies kann wie hier im Werk oder als einzelne Datei (z.B. im Anhang eines Werkes) erfolgen. Weitere Angaben sind nicht notwendig.

Kostenfreiheit
Dieses Werk, Bearbeitungen oder Übersetzungen und Werke, die auf diesem Werk beruhen, müssen unentgeltlich weitergegeben werden. Wird davon eine gedruckte Version erstellt, muss zumindest eine digitale Version dem gleichen Empfängerkreis ebenfalls kostenfrei zugänglich sein.

Datensparsamkeit
Erfolgt die Bereitstellung außerhalb einer öffentlichen Bildungseinrichtung oder einer Bildungseinrichtung in freier, nichtöffentlicher Trägerschaft, muss der Zugang dazu ohne Benutzerregistrierung, Erhebung von personenbezogenen Daten mit Ausnahme der technischen Verbindungsparameter oder einer Anforderung des Dokuments per Formular oder Mail (nötigenfalls durch Bereitstellung über einen weiteren Endpunkt) erfolgen.

Kontakt: mail@arminhanisch.de

WARUM

Ich mag keine Lernmittel, die beim Lesen stauben. Ich mag auch keine „Lehrbücher". Was ich dagegen mag, sind „Lernbücher", die Interesse wecken und mit deren Geschichten sich die Leserin identifizieren kann.

Geschichten, die Interesse wecken, authentisch sind und die Leserin dazu anregen, über die Dinge nachzudenken. Es sind Geschichten, die uns vom Lagerfeuer zum Mond gebracht haben. Du kannst niemand motivieren, Du kannst nur interessante Geschichten erzählen, die Fragen aufwerfen. Der Rest kommt von selbst. Dieser Text ist der Versuch einer solchen Geschichte.

VORWISSEN

Wer meinen Text „Zahlensysteme" gelesen hat, kann hier nahtlos anschließen. Die Geschichte von Alina und den nerdigen T-Shirt geht in diesem Text weiter und wir erfahren endlich, was die Binärzahlen darauf bedeuten.

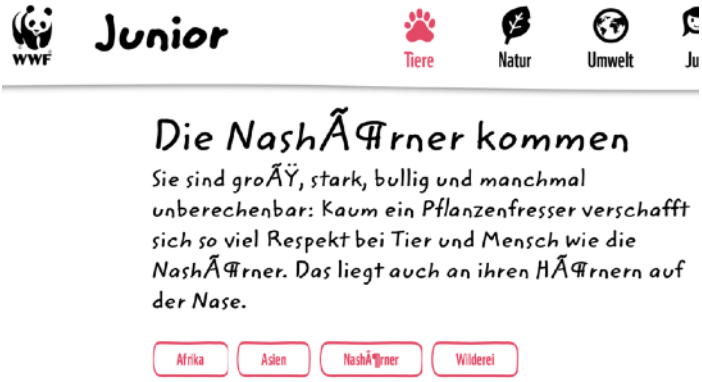
Für alle anderen: wer in der Lage ist, Zahlen aus dem Dezimalsystem in das Binärsystem und das Hexadezimalsystem umzurechnen, für den ist dieser Text kein Problem. Die Kenntnis von Begriffen wie Bit, Bytes und ein paar Stunden Erfahrung beim Surfen im Web sollten ebenfalls vorhanden sein.

KAPITEL 1

NASHÖRNER

„Papa… Papaaa!!“ Alina brüllte aus zwei Zimmern Entfernung quer durch die Wohnung. „Am Tablet geht was nicht, komm mal!“ Seufzend stand ich auf. Immerhin unterschied sich die detaillierte Art der Problembeschreibung nicht von dem, was ich von meiner Arbeit gewohnt war.

„Hallo, meine Tochter, hier ist Dein freundlicher IT-Support. Bitte legen Sie ein Ticket an und melden Sie sich dann wieder“, sagte ich und drehte mich grinsend wieder um. „Halt! Hier geblieben! Guck doch!“ Alina hielt mir den Bildschirm ihres Tablets entgegen, auf dem sie irgendwas für die Schule recherchierte.

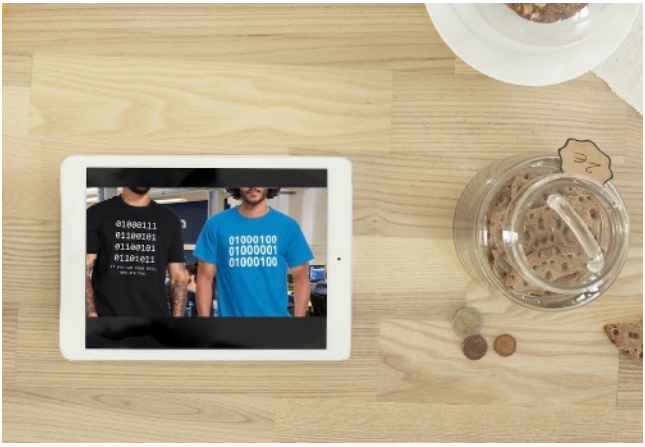


„Ah. Ein Encoding Fail, das haben wir gleich. Das muss UTF-8 sein“, meinte ich mehr zu mir selbst als zu meiner Tochter.

„What? Was ist Encoding und UTF und kannst Du das auch in Deutsch sagen, damit ich das selbst hinbekomme?“, sprudelte es aus Alina heraus. „Ich brauch das Nashorn-Zeug für mein Referat diese Woche!“

Ich beugte mich über das Notebook und klickte kurz rum. „So, jetzt sind die NashÃ¶rner wieder Nashörner. Vorschlag: Du machst das jetzt fertig und dann erkläre ich Dir, was da passiert ist. Das hat übrigens was mit den Binärzahlen auf den T-Shirts vom Wochenende zu tun. Wolltest Du ja eh wissen. Komm' ins Wohnzimmer, wenn Du soweit bist.“

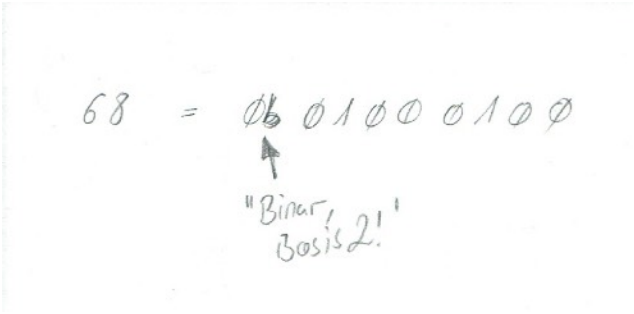
Ich wollte gerade das Zimmer verlassen, als meine Tochter schon hinter mir stand. „Ich hab die URL als Lesezeichen abgelegt, das Referat ist erst Ende der Woche. Du hast mir schließlich versprochen, dass Du mir erklärst, was auf den T-Shirts steht Los, dann erklär mal. Die Nashörner machen Pause.“



Meine Tochter, der Prokrastinations-Profi…

Alina legte mir das Foto von Hermann und mir in unseren Geek T-Shirts auf den Tisch und blickte mich an. „Also? Ich höre.“

Ich nahm einen Schluck Kaffee und einen Bleistift in die Hand. „Die 68 hast Du selbst ja ausgerechnet. Ich mal' das mal als Binärzahl hin. So."



„Übrigens, der Strich durch die Null ist was, dass viele Softwareleute machen, weil Du dann die Null nicht mit einem großen «Oh» verwechseln kannst", erklärte ich. „Es gibt sogar extra Schriftarten für Programmierer, die das am Rechner oder Smartphone auch machen. Sonst hast Du einem Ausdruck oder am Rechner echt Probleme, zu erkennen, ob das eine Null sein soll oder nicht. Hier ein Beispiel."

0O0O

■■

KAPITEL 2

CODES

„Ich hatte Dir ja schon erzählt, dass Computer keine «Zahlen» oder «Bilder» speichern. Alles, was ein Computer speichern kann, sind Bits. Wir Menschen brauchen aber auch Zeichen. Texte. Grafiken. Arbeitsblätter in BWR für meine Tochter", grinste ich. Alina hob warnend eine Augenbraue.

„Ok, Ok, wir müssen den Bits im Computer also eine Bedeutung geben. Aber wie macht man aus Bits Zeichen oder Bilder oder einen Song von BTS? Das ist wie mit Buchstaben. Wenn da «MIST» steht, sind die Informations-Stücke – die Buchstaben – für einen Engländer und einen Deutschen völlig gleich. Es ist die Bedeutung, die eine andere ist. Wir brauchen also eine Vereinbarung, welche Daten im Computer was bedeuten. Schon mal versucht, eine Daten-CD im CD-Spieler anzuhören. Dagegen ist Death Metal richtig melodisch.", schüttelte ich mich. Ich zeigte meiner Tochter ein Bild im Web. „Schau, das Problem ist uralt."

„Codes gab es schon lange vor den Computern. Wenn Du am Meer auf ein Schiff triffst, das diese beiden Flaggen oben am Mast hat, dann müssen sich alle einig sein, was diese bedeuten. Und zwar über alle Länder hinweg. Sonst meint der Eine, das bedeutet «*Mir fehlt Treibstoff*» und «*Wir brauchen Rum*» und der Nächste denkt, das wird bedeuten, dass sie gerade einen Schatz gefunden haben. In Wirklichkeit ist die echte Bedeutung «*Maschine aus, ich mache keine Fahrt*» und die darunter bedeutet «*Ich habe Taucher im Wasser, Vorsicht und Abstand halten*». Aber das ist nur eine Vereinbarung, ein Code. Ein Wert (die Flagge) wird durch einen anderen (die Bedeutung) ersetzt. Verwenden nicht beide Seiten die gleich Code-Tabelle, dann bekommst Du Deinen Nashorn-Salat."

„Lass uns einen Code bauen, Alina. Damit es einfach wird, bleiben wir bei Zeichen. Zuerst musst Du wissen, welche Zeichen Du überhaupt haben willst."

„Naja, alles halt. Was ich halt brauche, um Texte zu schreiben" kam als Antwort.

„OK, lassen wir das für den Anfang mal so. Dann noch die Ziffern und ein paar andere Zeichen wie Punkt, Komma, Fragezeichen, das Leerzeichen und so weiter", ergänzte ich. „Jetzt musst Du Dir für jedes Zeichen einen Wert ausdenken, denn der Computer dann binär speichert und all Deinen Freundinnen mitteilen, welcher Wert für welches Zeichen steht."

„Oh. Na, dann eben 1 für A, 2 für B, 3 für C und so weiter", erwiderte Alina. Sie begann, Zeichen aufzuzählen und aufzuschreiben. „Wie viele Zeichen hast Du gefunden?", fragte ich. Alina reichte mir einen Notizzettel mit einer Liste von Zeichen, Ziffern und Satzzeichen. „Das werden doch schnell viele, es sind ungefähr 60 Zeichen." Ich guckte fragend. „Wie viele Bits brauchen wir?"

Meine Tochter rechnete. Das heißt, sie tat das, was ich auch oft mache im Binärsystem: schnell die Zweierpotenzen aufzählen. „1, 2, 4, 8, 16, 32, 64 – halt, 64 brauchen wir nicht, also reichen 6 Bits. Stimmt's?"

Ich nickte zustimmend. „Ja, ich habe aus Deinem Zettel mal eine schöne Tabelle gebastelt. Ich habe übrigens bei 0 angefangen zu zählen, weil das ja auch ein Wert ist."

	0	1	2	3	4	5	6	7
0	A	I	Q	Y	6	?	$	;
8	B	J	R	Z	7	+	%	Ä
16	C	K	S	0	8	-	&	Ö
24	D	L	T	1	9	*	[	Ü
32	E	M	U	2	SP	/	]	
40	F	N	V	3	.	=	#	
48	G	O	W	4	,	(	_	
56	H	P	X	5	!	)	:	

„Du suchst das gewünschte Zeichen und zählst dann die beiden Zahlen der Beschriftungsachsen zusammen. Nehmen wir mal an, Du willst «HALLO!» übertragen…" Gemeinsam suchten wir die Liste zusammen.

```
H = 56
A = 0
L = 25
L = 25
O = 49
! = 60
```

„An den Empfänger sendest Du also die Werte 56, 0, 25, 25, 49, 60. Wenn der Empfänger die gleiche Tabelle hat, macht der das anders herum und bekommt zu den Werten die entsprechenden Zeichen heraus." Ich sah meine Tochter an.

„Das ist erstmal alles. Ein Code ist eine Vereinbarung, welcher Wert durch welchen anderen Wert ersetzt wird. Damit haben wir das Thema «Code» durch." Alina sah mich erstaunt an.

„Hä? Das ist alles? Aber die Zahl auf dem T-Shirt war doch 68, das ist ja schon mehr als mit 6Bit gehen! Komm schon, da fehlt doch noch was!"

„Ja und Nein. Ein Code ist wirklich nur das. Eine Vereinbarung zwischen Sender und Empfänger, welcher Code welchem Zeichen oder welcher Information entspricht. Das ist das «Nein», da fehlt nichts. Aber jetzt zum «Ja», was da schon noch fehlt. Es gibt in der Welt der Software und der Computer nicht nur **einen** Code. Was wir gerade gemacht haben, ist «*Alinas großartige 6Bit Tabelle*». Ich weiß aber nicht, ob das für Fame und virale TikToks ausreicht", lächelte ich.

„Dazu kommt noch, dass wir später über die **Darstellung** der Werte sprechen müssen. Das ist das, was als **Encoding** bezeichnet wird. Aber zuerst mal zurück zu den Binärzahlen, damit Du endlich raus bekommst, was auf den T-Shirts steht."

KAPITEL 3

ASCII

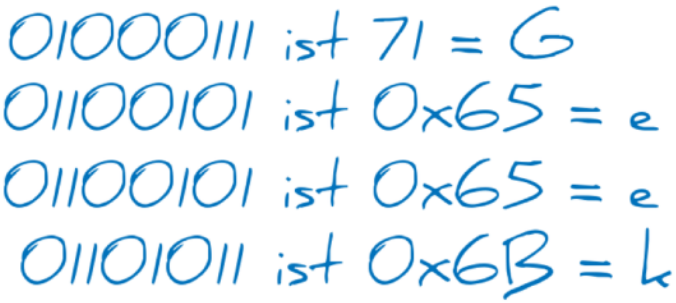
„Die Macher der T-Shirts nutzen einen anderen Code. Keine Angst, ich texte Dich jetzt nicht mit Dino-Kram zu, chill!“, sagte ich und zeigte Alina eine Website mit einer Tabelle.

	_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_	SPC	!	"	#	$	%	&	‘	(	)	*	+	,	-	.	/
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

„Es gibt eine Codierung, die sich seit der Mitte der 60er Jahre – sagte ich nicht, dass das Zeug so alt ist wie Dein Papa – durchgesetzt hat. Diese Tabelle wird als **ASCII** bezeichnet (*die Abkürzung wird als Wort ausgesprochen: «ass-kih»*). Die wird auch heute noch benutzt. Dazu gleich mehr. Aber jetzt kannst Du endlich das Rätsel der T-Shirts lösen.“

Meine Tochter schnappte sich das Tablet und nahm das Foto zur Hand. „Na endlich. War voll langweilig, der Theorie-Kram. So, mal sehen…“

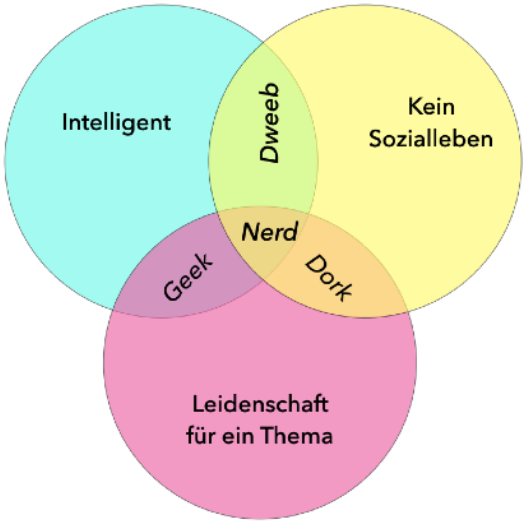
„Awww…" Alina grinste. „Papa, Du Ehrenmann. Kannst Du ruhig mal wieder anziehen, wenn es Dir nicht zu eng ist“, lachte sie. „So, jetzt der Hermann. Ist ja praktisch, dass Ihr da eine Zahl zweimal dabei habt.“



„Geek? Ich dachte, Nerd?“, wunderte Alina sich. „Was denn jetzt, ich dachte, sowas wie Ihr seid Nerds.“

Geeks & Nerds – ein kleiner Ausflug

„Nein, wir sind eigentlich Geeks, aber die Definitionen kennt kaum noch jemand. Wird immer verwechselt", entgegnete ich. „Hier, ein klitzekleiner Ausflug weg vom Thema für eine Minute". Ich zeigte Alina ein Diagramm.



„Das hat Scott Beale, der Gründer von *Laughing Squid* im Herbst 2009 veröffentlicht. Ich habe es übersetzt, da kannst Du selbst gucken, was jemand ist. Kannst morgen ja in der Schule mal rumfragen, wer das erklären kann."

„Aber zurück zum Bitkram". Ich legte das Diagramm wieder weg.

■■

„Dieser «*ASCII-Code*» kann 128 verschiedene Zeichen darstellen. Weil er 7Bit *breit* ist. Und in sieben Bit gehen – habe ich Dir ja schon mal erklärt – 2^7 verschiedene Werte rein. Und wenn Du die gleiche Codierung nutzt, dann kannst Du die übermittelten Daten auch lesen. Wenn Du eine andere hast, bei der das «A» zum Beispiel den Wert 80 hat, dann kommt da nur Unsinn raus."

```
Text "Hallo, Welt"
ASCII-Bytes:
0x48 0x61 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x65 0x6C 0x74 0x21
```

„Jetzt kommt jemand mit einem anderen Code mit einer anderen Zuordnung und versucht, das zu lesen. Ich nehme mal EBCDIC_0037. Ich habe Dir ja erklärt, dass es verschiedene Codes gibt. Dann kommt das raus."

```
Bytes von oben:
0x48 0x61 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x65 0x6C 0x74 0x21
Als EBCDIC decodiert: ç/%%?ïÁ%È
```

„Epic fail", sagte ich und zeigte Alina die Ausgabe auf dem Tablet. „So, jetzt zu Deinem «Nashorn-Problem». Mittlerweile ahnst Du bestimmt schon, was das Problem war, oder?"

Alina nickte. „Ja, das nennt sich – wie sagst Du immer – «shit in, shit out»". Sie grinste mich an. „Oh, ich wollte sagen, da gab es eine kleine Verwechslung bei der Codierung und Decodierung der Zeichen". Sie grinste mich an.

„Stimmt. In diesem Fall aber eher ein «good stuff in, wrong code, shit out", merkte ich an.

„Aber das ist doch echt voll kompliziert. Du musst immer wissen, dass alle den gleichen haben. Und warum gibt's dann überhaupt mehr als einen Code?"

KAPITEL 4

8BIT STATT ASCII

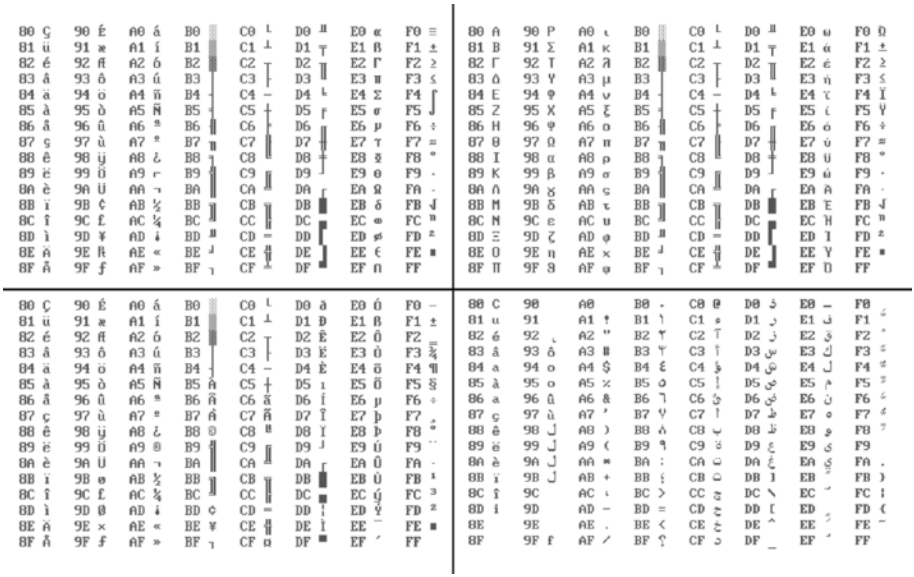
„Weil die Welt nicht ASCII spricht. Solange Du Englisch sprichst und in den USA bist, ist alles shiny. Das Ding heißt ja nicht umsonst «ASCII», das ist die Abkürzung für «*American Standard Code for Information Interchange*». Also «*amerikanischer Standardcode für den Austausch von Informationen*». Der Rest der Welt kam in der damaligen Computerwelt nicht vor bzw. nahm das, was die Amerikaner angeboten haben. Der Code hat nur 7 Bit, bei den 128 Zeichen ist Schluss."

Ich kritzelte auf einen Zettel: „Schau mal die deutsche Sprache an: «*Übel, übel, nörgelte der Räuber*». Fällt Dir was auf? Aber – und jetzt kommt die Computerversion des Turmbaus zu Babel – es wurden ja nur 7 Bit verwendet und in einem Byte stecken aber 8Bit. Man hatte also ein Bit übrig, mit dem herumgespielt werden konnte. Da bauten manche eine Fehlererkennung ein, andere markierten damit im Speicher Dinge wie das Ende eines Wortes oder das Bit blieb einfach leer. Wenn nun dieses eine Bit mehr für Zeichen genutzt werden kann, dann aber…"

Meine Tochter fiel mir ins Wort, „Ja cool, dann hast Du doppelt so viele Zeichen, weil eine Stelle mehr bei der Basis 2 bedeutet, dass sich die Zahl verdoppelt. Mit 7Bit hast Du 128 verschiedene Werte und mit 8Bit ja 256! Das reicht für alle Umlaute, oder? Haben das dann alle gemacht?"
Sie guckte mich neugierig an.

Ich seufzte. „Ja, und und alle gleichzeitig und ohne an die anderen zu denken. Jedes Land mit eigenen Zeichen und jeder, der irgendwelche Sonderzeichen haben wollte! Die haben einfach angefangen, diese neuen 128 Zeichenplätze mit den Werten 128 bis 255 so zu codieren, wie sie es für richtig hielten."

„Hier, ich zeige Dir mal vier verschiedene Beispiele für diesen 8Bit-Zeichensatz: IBM PC, Griechisch, Westeuropäisch und Arabisch. Wie sagst Du immer? «*Das ist cringe*»? **Das da** war mega-cringe! Hier, guck mal, was der Wert 178 (0xB2) bedeutet."



„Beim PC ist das so ein Rasterkästchen, bei den Griechen auch, bei Westeuropa auch. Aber bei Arabisch steht da ein «٢» – was'n das?"

„Das ist eine arabische Ziffer. Dieses Mal die echten arabischen Ziffern und bedeutet «2»", erklärte ich. "Das ist jetzt noch gut, da stimmen 3 von 4 Codes überein. Wenn Du Du also einen Klötzchenrahmen hast, sieht das zu 75% gut aus."

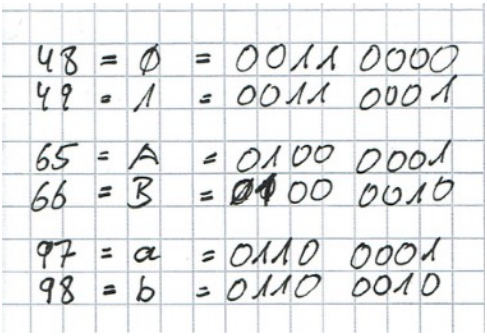
„Aber jetzt guck' Dir mal den Wert 245 (0xF5) an."

IBM PC = ⌠
Griechisch = Ÿ
Westeuropa = §
Arabisch = ﻵ

„Was? Da stimmt ja gar nix mehr. Das ist ein echter fail". Alina war entsetzt.

„Ja, deswegen war es früher so wichtig, dass immer klar war, welcher Code für gespeicherte Daten benutzt wird. Sonst gibt es nur Umlaut-Salat statt Text. Und irgendwann reichen Dir auch die 256 Zeichen nicht mehr. Denk mal an China und die Tausenden von Schriftzeichen. Und all die Emojis – no way bekommst Du das in 256 Zeichen."

Ich nahm ein Blatt und malte die Zahl 65,66, 48, 49, 97 und 98 auf ein Blatt, daneben die Binärzahlen dazu. „Um das zu verstehen, müssen wir kurz Bits gucken gehen." Alina seufzte. „Langsam wird mir klar, warum Ihr diese Nullen und Einsen sogar auf dem T-Shirt habt. Aber OK."



„Die Werte der einzelnen ASCII-Codes sind nämlich nicht Zufall, da haben die Leute damals echt nachgedacht. Guck' Dir mal die linken vier Bits an. Daran erkennst Du zum Beispiel, ob das ein Buchstabe ist oder nicht. Denn die Großbuchstaben haben alle das Bit für 2^6 (=64) gesetzt. Die vier Bit rechts sind dann die Position im Alphabet. Genau darum hat das «A» den Wert 65 – 64 für Buchstabe und 1 für eben A. Und «B» hat auch 64 plus 2, weil eben B der 2. Buchstabe im Alphabet ist. Alle Ziffern haben die beiden Bits für 2^5 und 2^4 gesetzt und die rechten vier Bits ist dann der Zahlenwert."

„Ja, cool!", staunte Alina. „Die haben da echt Hirn reingesteckt."

„Würde Dein Deutschlehrer anders formulieren, aber exakt."

Meine Tochter schmunzelte. „Leute, die alte Sprachen sprechen, sind meistens auch alt. Aber ich habe gerade rausgefunden, warum die kleinen Buchstaben alle mindestens 97 haben, das ist 96 und der Platz im Alphabet, also genau 32 mehr wie die Großbuchstaben. Yay!"

Ich nickte energisch. „Ja, Das Bit für 2^6 (=64) bedeutet «Buchstabe» und das Bit für 2^5 (=32) bedeutet «klein». Und wenn die linken vier Bits alle Null sind oder 0001, dann sind das sogenannte Steuerzeichen. Die zeigen nix an, die machen was wie zum Beispiel der Tabulator mit dem Code 0x9. Der Code 0xA (=10) bedeutet «LineFeed», also gehe eine Zeile weiter und so weiter. Ich erzähl Dir das ganze Zeug, weil Bits angucken wichtig ist, damit Du auf Deinem Smartphone Emojis verwenden kannst. Ist also keine Luftnummer zum Büffeln, sondern ohne Bits gucken gibts keinen Smiley 😄 und keinen Pile of poo 💩."

Ich blinzelte meiner Tochter verschwörerisch zu. „Aber es gibt eine Lösung. Doch zuerst ein Eis als Nervennahrung." Sie war schneller an der Tür als ich aufstehen konnte. 😄

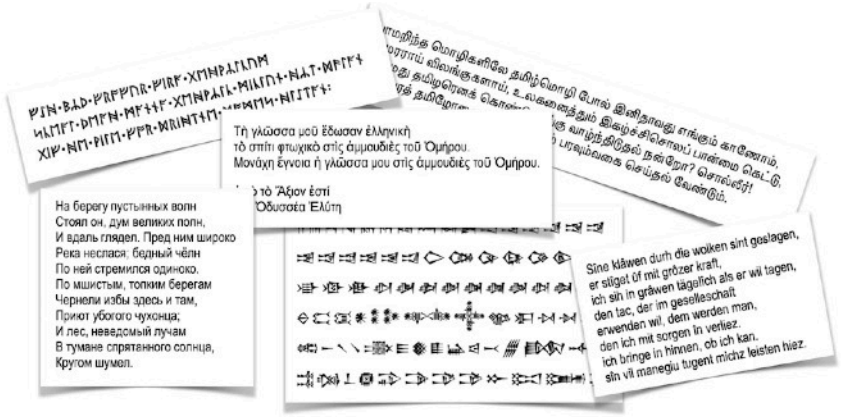
KAPITEL 5

ZEICHEN FÜR DIE WELT

Ich schleckte an meinem Pistazieneis und sagte „ASCII war eine Lösung für eine englischsprachige Welt und auch da reichten 256 Zeichen schon nicht aus. Es gibt allerdings zwei Organisationen, die schon sehr früh in der Zeit der persönlichen Computer eine Lösung dafür suchten. Eine davon war Apple und die andere Xerox. Apple war ab Mitte der 80er Jahre die einzige Firma, die mit dem Mac und dem Laserdrucker ein System anbot, mit dem Du Schriftsatz auf dem PC machen konntest. Damit konnten Leute auch verschiedene Schriftarten und verschiedene Alphabete nutzen. War aber echt ein Mordsaufwand. Und Xerox hatte in Palo Alto in Kalifornien ein Forschungszentrum. Dort wurde so ziemlich alles erfunden, was auch heute noch wichtig ist: vom Ethernet-Netzwerk, mit dem das Internet läuft, über den Laserdrucker bis zur grafischen Benutzeroberfläche und sogar die Computermaus."

„Steve Jobs von Apple besuchte Ende 1979 zusammen mit einigen Mitarbeitern dieses Forschungszentrum, wurde herumgeführt und er ließ sich «inspirieren». Erfunden in Palo Alto, verkaufsfähig gemacht und vermarktet in Cupertino, sozusagen. Aber genug der Geschichte. Nur soviel, Leute aus den beiden Firmen haben sich 1987 und 1988 zusammen gesetzt und eine Art «neues, weltweit nutzbares ASCII» entwickelt. Klingt aber blöd. Also haben sie das Baby «**Unicode**» getauft. Als Kurzwort für eine wirklich eindeutige, universell nutzbare Definition aller Zeichen: «*Universal Coded Character Set*»"

„Oha. Wirklich alle?", fragte Alina zwischen zwei Happen Eis.



„Ja, es gibt sogar Keilschrift und Linear B im Unicode-Standard, Tausende von Jahren alt. Und isländische Runen und Thai und Hebräisch und Blindenschrift. Wenn Du ein System bauen willst, das die ganze Welt benutzen kann, dann muss das so erweiterbar sein, dass wir wirklich alle benötigten Zeichen der Welt darin abbilden können. Es gibt den Unicode-Witz, dass Klingonisch nur deshalb nicht im Standard enthalten ist, weil kein Klingone zu den Meetings erschienen ist. Ich finde den witzig", grinste ich breit.

„Da ich weiß, dass Eure Generation lieber Videos guckt, als Textwüsten zu durchwandern, erzähle ich da auch nicht mehr. Zu Unicode gibt es ein nettes 5-Minuten-Video auf YouTube und ja, es kommen jede Menge Emoji drin vor. Hier is der Link: `https://www.youtube.com/watch?v=-n2nlPHEMG8` oder Du scannst einfach meinen QR-Code zum Video". Ich drehte Alina das Smartphone zu. „Guck ruhig, dann kann ich inzwischen mein Eis fertig essen."

Ich blickte auf, als Alina das Smartphone wieder weg legte. „Also: ASCII ist ein Code. Welcher Wert für welches Zeichen steht. Der Unicode-Standard ist das auch. Aber er sagt nichts darüber aus, wie viele Bytes Du nehmen sollst oder wie das binär aussieht. Der Unicode-Standard definiert erst einmal nur eine eindeutige Codenummer für jedes Zeichen, das nennt sich dann «Codepoint». Geschrieben wird das als großes U, gefolgt von einem Pluszeichen und dann dem Codewert als Hexadezimalzahl, z.B. `U+0041`."

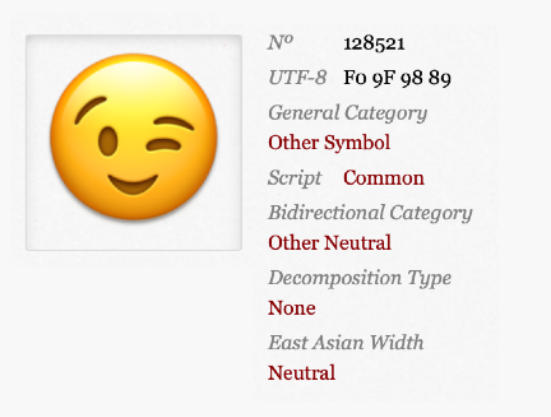
„`U+0041` ist also das Gleiche wie `0x41`, aber das «U+» sagt, dass das ein Unicode-Codepoint ist und dass das dezimal 65 sind. Hey, das ist das große «A», das weiß ich mittlerweile schon auswendig."

„Alina Alpha-Checkerin“, sagte ich stolz und lächelte. „Jetzt kommt die erste Überraschung. Wir brauchen mittlerweile etwas mehr als Deine 60 Zeichen mit «*Alinas großartige 6Bit Tabelle*» von vorhin. Es gibt ja schon so viele Emoji. Im aktuellen Unicode-Standard sind über 140.000 Zeichen enthalten! Dazu kommt: wenn ein Zeichen einmal in den Unicode-Standard aufgenommen wurde, dann bekommt es diese weltweit eindeutige Nummer, den Codepoint. Der wird nie wieder aus dem Unicode-Standard entfernt. Daher ist das Unicode-Konsortium sehr, sehr vorsichtig, wenn jemand ein neues Zeichen oder ein neues Alphabet im Standard haben haben will. Denk' an die Klingonen“, hob ich grinsend den Zeigefinger.

„Was aber genial ist, dass die Entwickler von Unicode für alle Zeichen, die im ASCII-Zeichensatz vorkommen, exakt die gleichen Nummer vergeben hat. Das «A» hat immer noch den Code 65, die «0» immer noch die 48, während ein asiatisches Zeichen vielleicht 58776 hat. Daher ist für ASCII-Text kein Unterschied zu Unicode. Damit kannst Du all die alten Computer und Programme, die nur ASCII verstehen, immer noch mit Daten füttern. Und weil das mit dem ewigen «A» langsam echt öde wird, nehmen wir das Zwinker-Smiley dazu, ok?“

„Stimmt, das muss ja auch einen eigenen Codepoint haben“, rief Alina.

„Hat es“, zeigte ich auf die Binärzahl aus meiner App mit dem Programmier-Rechner. „Das ist `U+1F609`. Das ist ja hexadezimal, in dezimal ist das 128521. Jeder Unicode Codepoint hat sogar einen offiziellen Namen: *WINKING FACE* in diesem Fall. Und alle Unicode Codepoints sind in «Blöcken» organisiert, die meistens ein Alphabet, eine Gruppe von zusammengehötigen Zeichen umfasst. Der Zwinker-Smiley liegt im Block «Emoticons».“



„Damit haben wir den Code. Also der Wert, der für das Zeichen steht. Jetzt kommt endlich das «Encoding»“.

KAPITEL 6

WAS IST EIN ENCODING?

Ich machte ein wichtiges Gesicht. „Ein Encoding ist die Art und Weise, wie der Wert eines Codes übermittelt wird, also die Darstellung." Ich konnte sehen, wie meine Tochter die Augen zusammen kniff. Da musste ein Beispiel her.

„Klingt Bananas? Kein Problem, ich habe was für Dich". Ich zog meine Geldbörse heraus und nahm einen 5€-Schein und legte den auf den Tisch. „Liegen lassen, Tochter, das ist ein Lehrmittel", scherzte ich.

„So, der Wert ist also 5. Das kann ich übermitteln oder beschreiben als eines von diesen hier". Ich schnappte mir eine dieser hauchdünnen Eis-Servietten und versuchte, darauf etwas aufzuschreiben. Nach zwei durchgerissenen Servietten stand Alina auf, ging zur Theke und kam mit einem Schreibblock zurück. „Hardware ist echt nicht so Dein Ding, Papa, oder?" Grinsend reichte sie mir den Block. „Haha. Sehr witzig", grummelte ich und begann zu schreiben.

Fünf

5

Five

IIIII

„Das alles transportiert die Information «5», aber in unterschiedlichen «Encodings». Ein «Encoding» ist also die Art und Weise, wie Du einen Wert verpackst, wenn man es so formulieren möchte."

Alina zeigt auf die Skizze. „Ah OK, aber was ist dieser Knubbel da neben der Zahl?"

„Das ist auch eine Fünf, aber nicht in lateinischen, sondern in arabischen Ziffern. Ich habe noch einen 5 Dinar-Schein aus Jordanien, da sieht das aus wie eine Null bei uns. Immer gut für einen Gag", grinste ich. „Habe ich das einigermaßen verständlich erklärt, das mit dem Unterschied zwischen einem Wert und seinem Encoding?"

Ich erntete ein Nicken von meiner Tochter. „Bis jetzt denke ich, dass ich das verstanden habe. Aber wie kommt ich jetzt von diesem Codepoint zu den Bits im Computer?"

„Ja, wir brauchen ein Encoding für den Codepoint. Also wie stellen wir 128521 so dar, dass wir das in Bytes und Bits angeben können", nickte ich.

„Es gibt zur Zeit über 140000 Zeichen, da kommen bestimmt noch ein paar dazu. Da mit 16 Bit nur 65536 Werte möglich sind (weil 2^{16}, *2 mögliche Werte pro Stelle und 16 Stellen*), brauchen wir mindestens drei Byte", rechnete ich.

„Dann nimmt man halt für jedes Zeichen einfach vier Byte, das reicht die nächsten Jahre." Alina war da ganz schnell fertig. „Du sagst doch immer, Speicher kostet nichts mehr."

„Ja, heute. Aber Unicode stammt aus einer Zeit Mitte bis Ende der 80er, also vor vierzig Jahren, meine Liebe. Wenn ich für jeden Codepoint vier Bytes nehme, also einfach eine Art «*aufgeblasenes ASCII*», dann verschwende ich eine **Menge** Platz", erwiderte ich. „Überleg' mal, ein «A» mit dem Code 65 sieht dann so aus."

```
0000 0000   0000 0000   0000 0000   0100 0001
```

Ich zeigt auf die lange Binärzahl. „Das sind echt viele Nullen für ein paar Einser ganz rechts. Das letzte Byte würde reichen. Wenn wir jetzt normalen englischen Text speichern, dann wäre jede Textdatei auf einmal vier Mal so groß wie vorher, weil wir statt einem Byte plötzlich vier Byte verbraten."

Ich zeigte auf Alinas Smartphone. „Nur so als Beispiel: Dein erstes Smartphone hatte 16 Gigabyte Speicher. Das sind 1024 Mal 16 Megabyte. Und ein Modul Speicher mit 16 Megabyte Speicher kostete Ende der 80er noch ungefähr 1000 Mark, also etwa 500 Euro. Da hätte Dein Handy nur für den Speicher schon mehr als eine halbe Million Euro gekostet!"

Alinas Reaktion war ziemlich eindeutig:

„Ohje, damals mussten die armen Leute wirklich Speicher sparen. Voll old school!"

„Es gibt aber tatsächlich ein Encoding, das für jedes Zeichen einfach 32 Bits überträgt, so wie oben in dem Beispiel. Der Name dafür ist «UTF-32». Das ist natürlich wieder so eine nerdige Abkürzung und steht für «*UCS Transformation Format with 32 bits*». Das UCS ist auch wieder eine Abkürzung für «*Universal Coded Character Set*», also Unicode. In verständlich also «*Unicode Transformationsformat mit 32 Bit*»."

„Der Haken an UTF-32 fällt auf, wenn wir uns nochmal die Bits angucken". Ich malte die Bits aus der App auf den Zettel.

„Hier, der Text «A 😉» im Encoding UTF-32."

```
0000 0000    0000 0000    0000 0000    0100 0001
0000 0000    0000 0001    1111 0110    0000 1001
```

„Von acht Byte, die wir für die Übermittlung benötigen, sind bei vier alle Bits auf Null. Wir haben also gerade 50% unserer Übermittlungsgeschwindigkeit oder Bandbreite für Nichts verblasen. Und wie Du beim YouTube streamen immer meckerst, ist Bandbreite nicht ganz unwichtig. Wir hätten die Daten eigentlich doppelt so schnell übertragen können oder nur die Hälfte des Speicherplatzes gebraucht."

Alina seufzte, „Aber was machen wir dann? Dann müsste man doch bei jedem Zeichen wissen, wie viele Bytes das braucht und das ist ja noch mehr Information, die Platz braucht."

„Aber Du bist auf dem richtigen Weg", stimmte ich zu. „Deswegen habe ich Dich vorhin mit der Bitfummelei so genervt. Es hatten ein paar kluge Köpfe eine geniale Idee und diese geniale Idee heißt «**UTF-8**»."

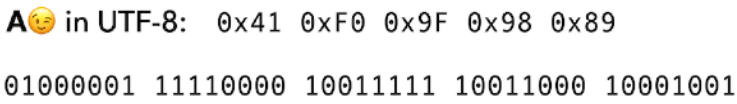
KAPITEL 7

DAS WEB SPRICHT UTF-8

Ich wedelte stolz mit einer übrig gebliebenen Serviette. „Und als historische Anekdote: der Entwurf für UTF-8 entstand im September 1992 in einem Restaurant in Palo Alto auf einer Platzserviette!“

„Der geniale Trick bei UTF-8 ist, dass die ersten 128 Zeichen auch nur ein Byte brauchen. Problem 1 ist damit gelöst. Und weil die Codewerte gleich sind, können auch alte Museumsrechner und alte Software, die nur Bytes und ASCII kennt, auch damit arbeiten. Und wenn ein Zeichen kommt, das nicht ASCII ist, wird immerhin etwas ausgegeben, aber die Verarbeitung bricht nicht ab. Das waren die seltsamen Zeichen bei Deinem Nashorn-Problem. Problem 2 gelöst. Und für höhere Code-Points werden dann einfach zwischen zwei und sechs Byte gebraucht. Die folgenden 1920 Zeichen benötigen zwei Byte. Das sind genialerweise fast alle häufig genutzten Zeichen wie Griechisch, Kyrillisch, Koptisch, Arabisch, Hebräisch und mehr. Drei Byte brauchen asiatische Zeichensätze wie Japanisch, Chinesisch, Koreanisch. Alle anderen benötigen zwischen vier und sechs Byte. Damit sparst Du bei den am häufigsten genutzten Zeichen am meisten Platz gegenüber UTF-32 und asiatische Zeichen brauchten auch vorher schon mehr als ein Byte. Auch Problem 3 gelöst.“

Ich tippte den Text "A😉" auf einer Website zum Konvertieren von und nach UTF-8 ein und zeigte Alina das Ergebnis.



„Siehst Du? Statt 8 Byte haben wir nur 5 benötigt. Über ein Drittel eingespart."

„Damit hatte die Computerwelt eine Möglichkeit, alle Schriftzeichen der Welt in einem Code zu haben, dem Unicode-Standard. Eine eindeutige Nummer für jedes Zeichen. Dazu kommt mit UTF-8 ein Encoding, also eine Darstellung in einer Form, die nicht zu viel Platz verschwendet, kompatibel mit dem alten Zeug und ASCII ist, genial einfach designed ist und tatsächlich in den letzten Jahren **die** vorherrschende Form des Encodings, der Darstellung von Unicode-Codepoints geworden ist. Über 95% aller Webseiten der Welt nutzen UTF-8, bei vielen Sprachen sind es 100%. Es gibt keine Probleme mehr beim Übertragen von Texten zwischen Systemen, jedes Zeichen hat einen eindeutigen Namen und im Unicode-Standard sind noch zusätzliche beschreibende Daten zu jedem Codepoint hinterlegt. Zum Beispiel, ob das eine Ziffer ist oder ein Kleinbuchstabe oder etwas zusammengesetztes wie das «Ä». UTF-8 ist für mich der genialste Hack in der Geschichte der Textspeicherung auf dem Computer!"

Alina lehnte sich mit weit offenen Augen zurück. „Papa, Du solltest Verkäufer werden. Bekommst Du Kohle von den Unicode-Leuten?" Sie lachte los.

Ich grinste zurück. „Das wäre schön, nichts gibt's. Aber jetzt weißt Du, warum Deine Nashörner plötzlich zu NashÃ¶rnern wurden. Der Webserver hatte die Seit als UTF-8 ausgeliefert, aber Dein Browser war irgendwie der Meinung, dass es sich um ASCII oder was anderes gehandelt hat. Aus diesem Grund kannst Du bei fast allen Browsern im Menü auch einstellen, welches Encoding der Browser verwenden soll, falls es mal hakt."

Ich winkte dem Kellner, um zu bezahlen. „So, das Eis ist weg, Du weißt jetzt, dass ein Code eine Zuordnung ist, welcher Wert durch welchen anderen Wert ersetzt werden soll wie das «A» durch den Code «65» und dass ein «Encoding» die Art und Weise der Darstellung dieses Codes ist."

„Und das ganz einfacher Text nie ganz einfacher Text ist, sondern dass es immer darauf ankommt, was Du aus den Bits im Computer machst."

Ich stand auf. „Jetzt musst Du wohl oder übel zurück nach Afrika zu den Nashörnern und Dein Referat fertig machen. Dieses Mal sogar mit «Ö»."

ÜBER DEN AUTOR

Ich kam zu Beginn der 80er Jahre über den Amateurfunk und ein paar Computer in der Schule zur Arbeit mit dem Computer. Seitdem war ich als System- und Netzwerkadministrator, Supportleiter, Systemanalytiker, Berater, Entwickler, IT-Trainer, CRM-Consultant, Projektleiter und Systemarchitekt tätig.

Seit vielen Jahren engagiere ich mich im Bildungsbereich. In meiner Freizeit schreibe und fotografiere ich, arbeite an Open Source Projekten mit und sammle seltene Single Malts. Auf Twitter bin ich @derLinkshaender.

BILDNACHWEIS

Alle Bilder sind entweder frei und ohne Angabe nutzbare Bilder von pixabay.de oder stammen aus dem Privatportfolio des Autors.