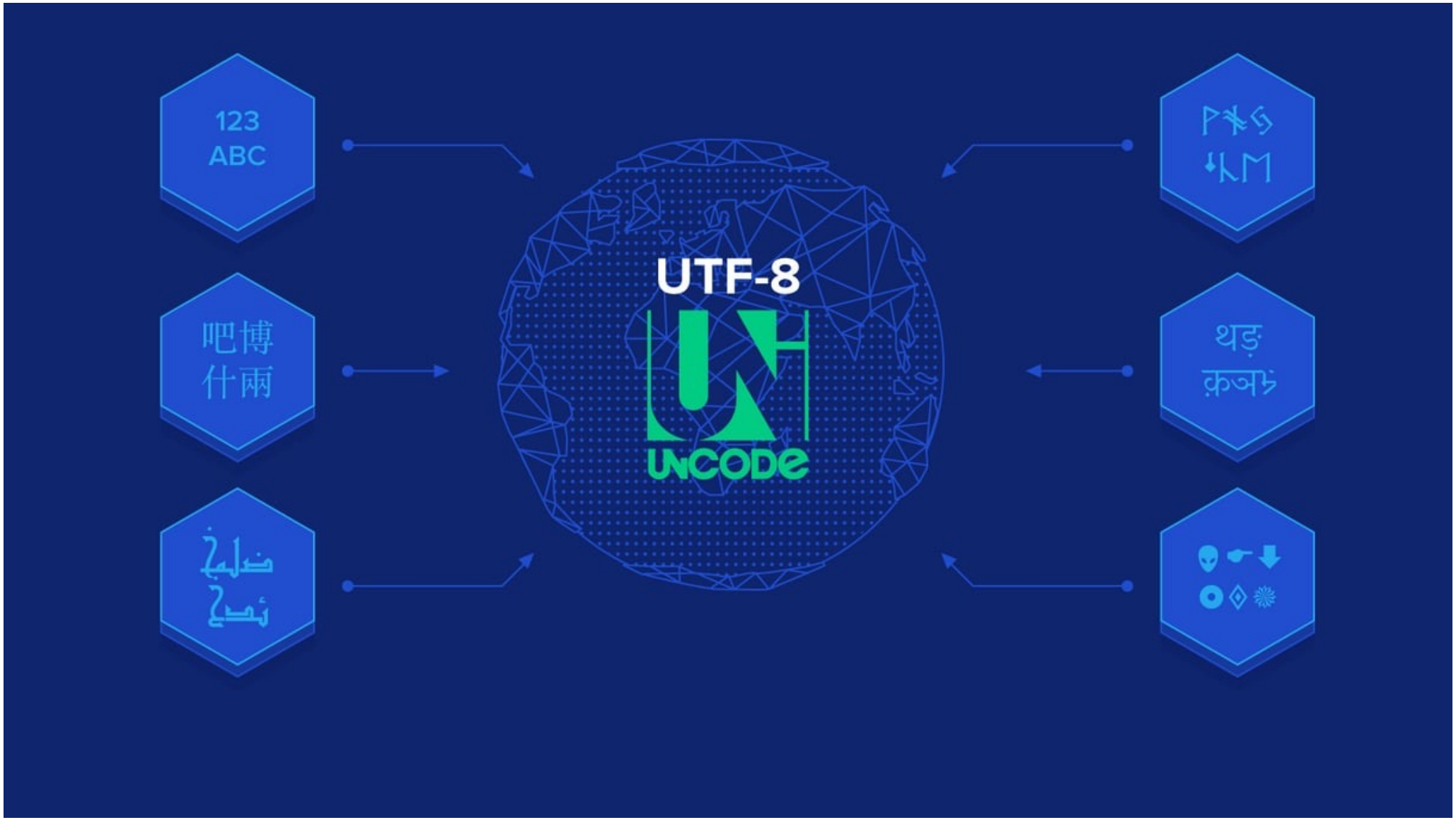
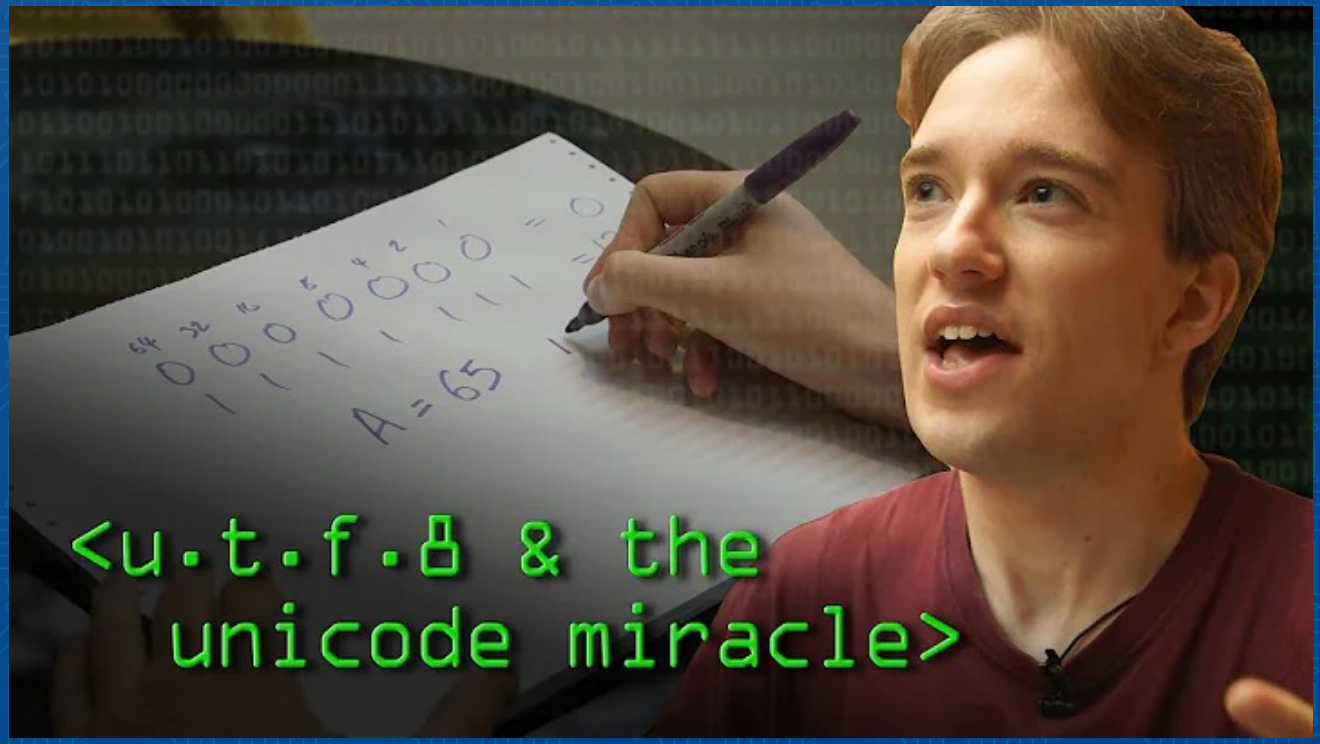
A Practical Guide to UTF-8 and Character Encodings

Utah Stack JS meetup

Ken Snyder


123
ABC
吧博
什兩
UTF-8
UNICODE

Tom Scott: “The most beautiful hack”

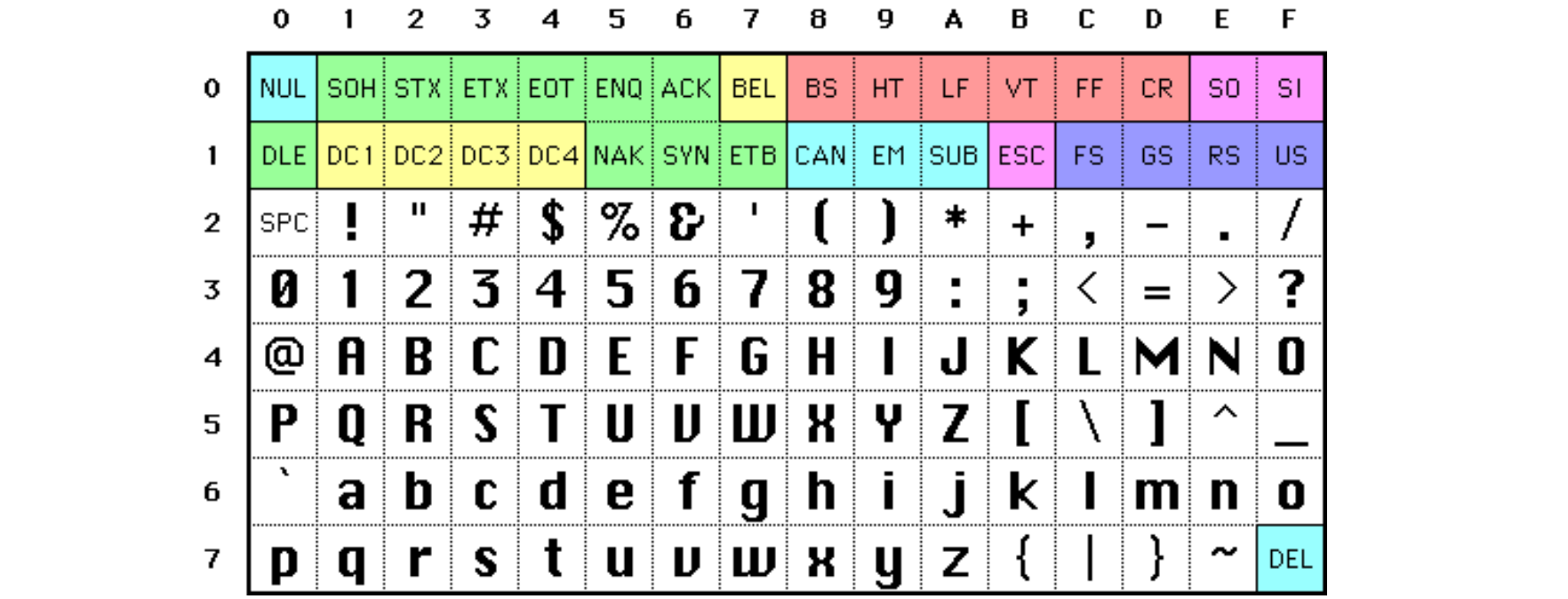


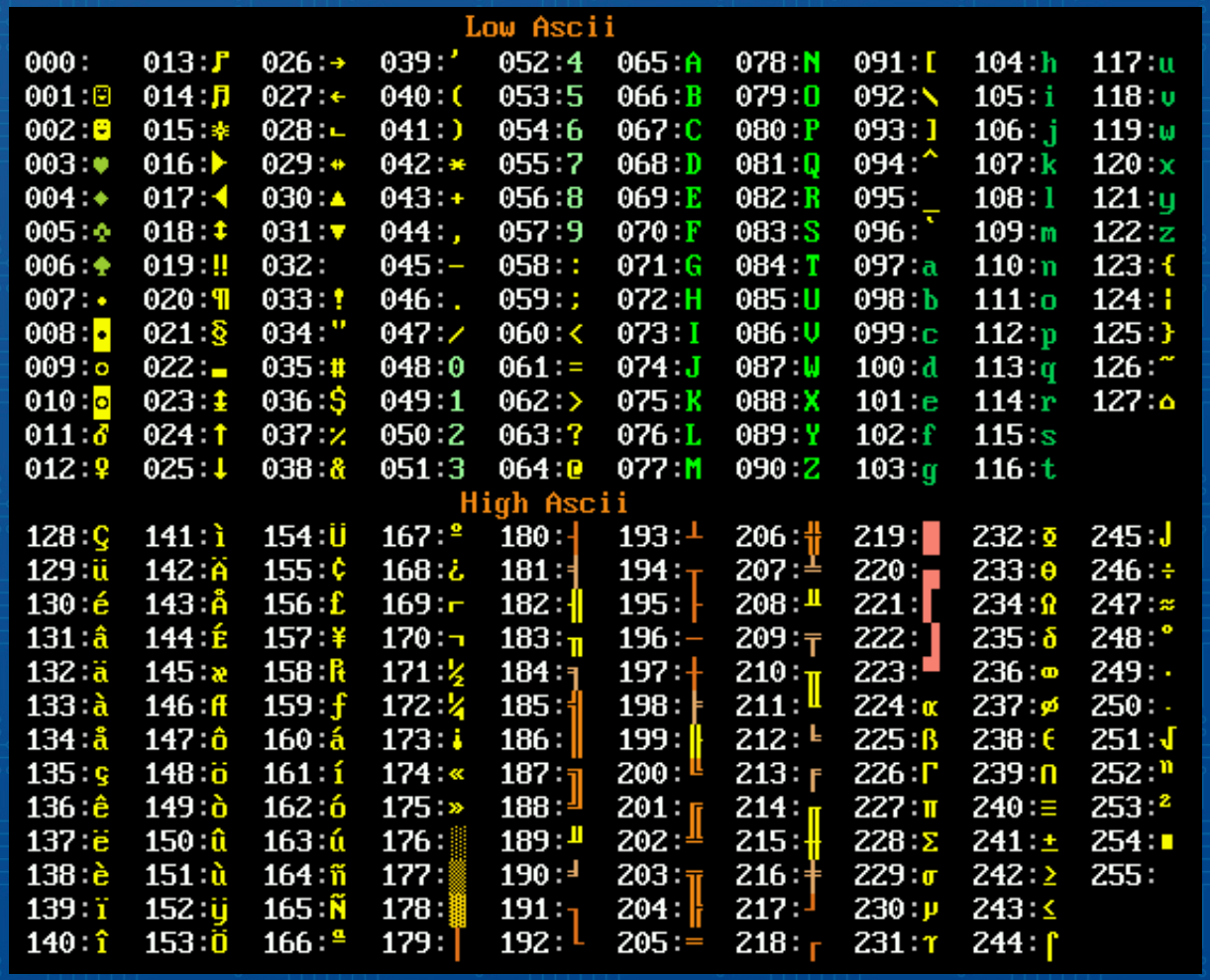
What we'll cover

- History
- Design
- Landmines
- Web technologies
- Tools
- Emoji

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com


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


Low Ascii
000:
001:☺
002:☻
003:♥
004:♦
005:♣
006:♠
007:•
008:◘
009:○
010:◙
011:♂
012:♀
013:♪
014:♫
015:☼
016:►
017:◄
018:↕
019:‼
020:¶
021:§
022:▬
023:↨
024:↑
025:↓
026:→
027:←
028:∟
029:↔
030:▲
031:▼
032:
033:!
034:"
035:#
036:$
037:%
038:&
039:'
040:(
041:)
042:*
043:+
044:,
045:-
046:.
047:/
048:0
049:1
050:2
051:3
052:4
053:5
054:6
055:7
056:8
057:9
058::
059:;
060:<
061:=
062:>
063:?
064:@
065:A
066:B
067:C
068:D
069:E
070:F
071:G
072:H
073:I
074:J
075:K
076:L
077:M
078:N
079:O
080:P
081:Q
082:R
083:S
084:T
085:U
086:V
087:W
088:X
089:Y
090:Z
091:[
092:\
093:]
094:^
095:_
096:`
097:a
098:b
099:c
100:d
101:e
102:f
103:g
104:h
105:i
106:j
107:k
108:l
109:m
110:n
111:o
112:p
113:q
114:r
115:s
116:t
117:u
118:v
119:w
120:x
121:y
122:z
123:{
124:|
125:}
126:~
127:⌂
High Ascii
128:Ç
129:ü
130:é
131:â
132:ä
133:à
134:å
135:ç
136:ê
137:ë
138:è
139:ï
140:î
141:ì
142:Ä
143:Å
144:É
145:æ
146:Æ
147:ô
148:ö
149:ò
150:û
151:ù
152:ÿ
153:Ö
154:Ü
155:¢
156:£
157:¥
158:₧
159:ƒ
160:á
161:í
162:ó
163:ú
164:ñ
165:Ñ
166:ª
167:º
168:¿
169:⌐
170:¬
171:½
172:¼
173:¡
174:«
175:»
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:α
225:ß
226:Γ
227:π
228:Σ
229:σ
230:µ
231:τ
232:Φ
233:Θ
234:Ω
235:δ
236:∞
237:ø
238:ϵ
239:∩
240:≡
241:±
242:≥
243:≤
244:⌠
245:⌡
246:÷
247:≈
248:°
249:∙
250:·
251:√
252:ⁿ
253:²
254:■
255:

Problem: No interoperability

Common character encodings [edit]

- ISO 646
  - ASCII
- EBCDIC
- ISO 8859:
  - ISO 8859-1 Western Europe
  - ISO 8859-2 Western and Central Europe
  - ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
  - ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
  - ISO 8859-5 Cyrillic alphabet
  - ISO 8859-6 Arabic
  - ISO 8859-7 Greek
  - ISO 8859-8 Hebrew
  - ISO 8859-9 Western Europe with amended Turkish character set
  - ISO 8859-10 Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
  - ISO 8859-11 Thai
  - ISO 8859-13 Baltic languages plus Polish
  - ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
  - ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
  - ISO 8859-16 Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861,
- Mac OS Roman
- KOI8-R, KOI8-U, KOI7
- MIK
- ISCII
- TSCII
- VISCII
- JIS X 0208 is a widely deployed standard for Japanese character encoding that has several encoding forms.
  - Shift JIS (Microsoft Code page 932 is a dialect of Shift_JIS)
  - EUC-JP
  - ISO-2022-JP
- JIS X 0213 is an extended version of JIS X 0208.
  - Shift_JIS-2004
  - EUC-JIS-2004
  - ISO-2022-JP-2004
- Chinese Guobiao
  - GB 2312
  - GBK (Microsoft Code page 936)
  - GB 18030
- Taiwan Big5 (a more famous variant is Microsoft Code page 950)
  - Hong Kong HKSCS
- Korean
  - KS X 1001 is a Korean double-byte character encoding standard
  - EUC-KR
  - ISO-2022-KR

UTF-8 vs utf-8 vs utf8

- UTF-8 is correct
- utf-8 is okay in case-insensitive situations
- utf8 works in most places
- Including Node, HTTP and HTML

Requirements

1. English text must be one byte per character
2. Can’t have 8 zeros in a row (NULL terminator)
3. Backwards compatible with ASCII
4. Easily traversable/navigable

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Usable Bits	Max Possible Characters
U+0000	U+007F	0xxxxxxx				7	128
U+0080	U+07FF	110xxxxx	10xxxxxx			11	1,920
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx		16	61,440
U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	21	1,048,576

Will theoretically work to 6 bytes

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Usable Bits	Max Possible Characters
U+0000	U+007F	0xxxxxxx						7	128
U+0080	U+07FF	110xxxxx	10xxxxxx					11	1,920
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx				16	61,440
U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx			21	1,048,576
U+200000	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx		26	67,108,864
U+4000000	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	31	2,147,483,648

Navigation in a string

Starts with a zero: 1 byte ASCII character = a U+0061 or 97

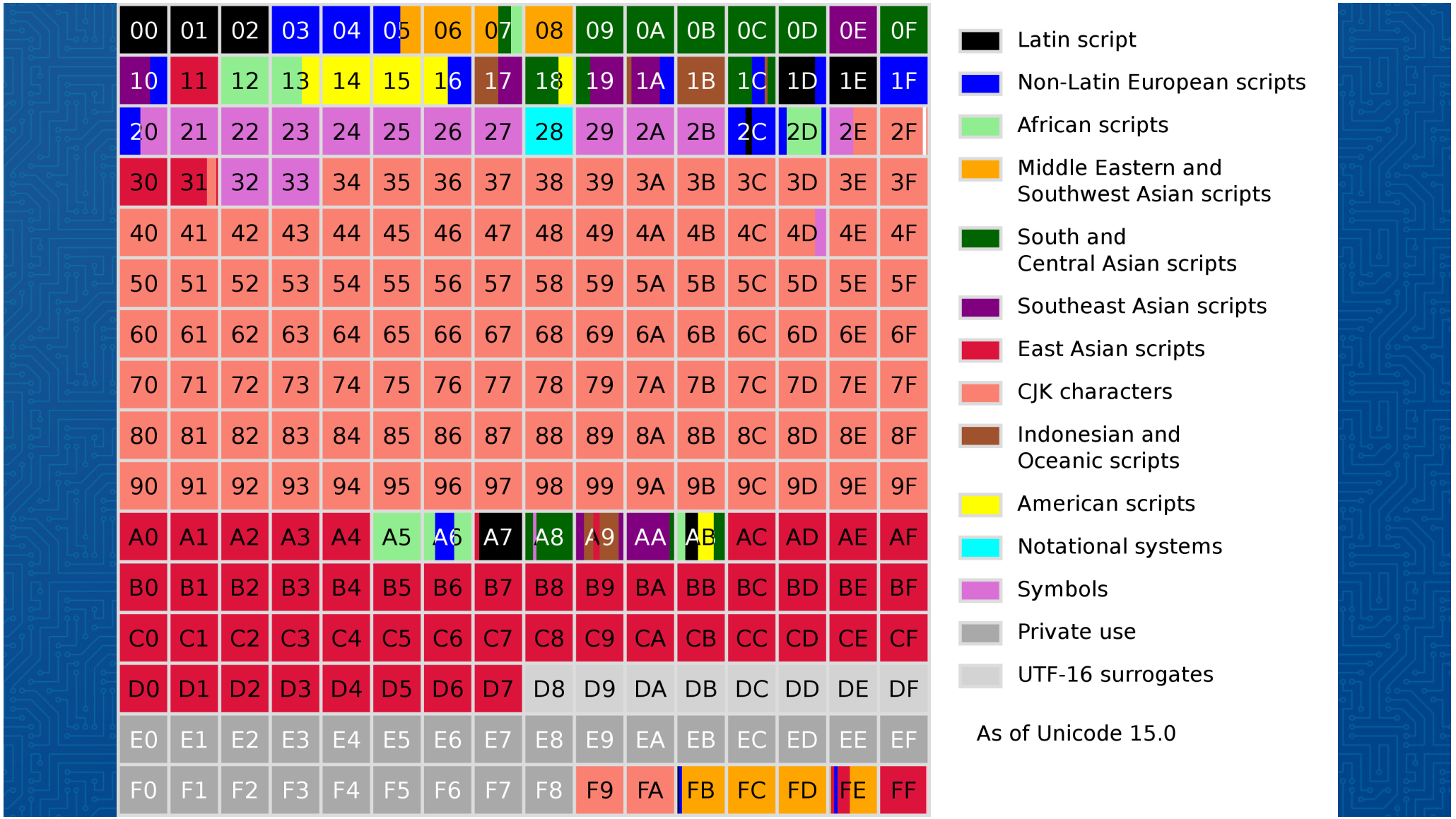
Starts with 3 ones: 3 bytes = ✈️ U+2708 or 14,851,208

Starts with 4 ones: 4 bytes = 🚀 U+1F680 or 4,036,991,616

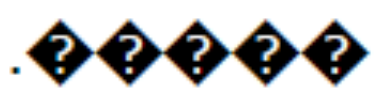
```
01100001 11100010 10011100 10001000 11101111 10111000
10001111 11110000 10011111 10011010 10000000 11110000
10011111 10010000 10111000 11000011 10100110 00001010
```

Growth and evolution over time

Year	Version	Total Assigned	Emoji Count	Max Bytes
1991	v1.0	12,795	0	3
1993	v1.1	40,635	0	3
1996	v2.0	178,500	0	4
1998	V2.1	178,502	0	4
1999	v3.0	188,809	0	4
2001	v3.1	233,787	0	4
2003	v4.0	236,029	0	4
2006	v5.0	238,671	0	4
2010	v6.0	249,031	0	4
2014	v7.0	252,603	0	4
2015	v8.0	260,319	0	4
2016	v9.0	267,819	1,788	4
2017	v10.0	276,337	2,666	4
2018	v11.0	277,021	2,789	4
2019	v12.0	277,575	3,019	4
2020	v13.0	283,506	3,304	4
2021	v14.0	284,344	3,633	4
2022	v15.0	288,833	3,664	4


00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Latin script
Non-Latin European scripts
African scripts
Middle Eastern and Southwest Asian scripts
South and Central Asian scripts
Southeast Asian scripts
East Asian scripts
CJK characters
Indonesian and Oceanic scripts
American scripts
Notational systems
Symbols
Private use
UTF-16 surrogates
As of Unicode 15.0

Mojibake

C	D
. 	UTF-8
.½¿)½¿)½¿)½¿)	Windows-1255
붛뾽붛뾽붛뾽	UTF-16
וַיְדַבֵּר	Passed with function

æ–‡å—åŒ–ã ‘

å‡ºå…¸: ãƒ•ãƒªãƒ¼ç™¾ç§‘äº‹å…¸ã€Žã,¦ã,£ã,ãƒšãƒ‡ã,£ã,¢ï¼ˆWikipediaï¼‰ã€

W ã,¦ã,£ã,ãƒšãƒ‡ã,£ã,¢ã §ã ®æ–‡å—åŒ–ã ‘ã «ã ¤ã „ã ¦ã ¯ã€ Help:ç‰¹æ®Šæ–‡å—ã,’ã ”è¦§ã ã ã •ã „ã€,

æ–‡å—åŒ–ã ‘ï¼ˆã,,ã ¯ã °ã ‘ï¼‰ã ¨ã ¯ã€ ã,³ãƒ³ãƒ”ãƒ¥ãƒ¼ã,¿ã §æ–‡å—ã,’è¡¨ç¤ºã ™ã,‹éš›ã «ã€ æ£ã —ã è¡¨ç¤ºã •ã,Œã ªã „ç ¾è±¡ã ®ã “ã ¨ã€,

- ä¾‹ï¼šã€Œæ–‡å—åŒ–ã ‘ã € ã Œã€ ã€Œ Ã¦â€“‡â€¡Ã¥Âå€”Ã¥Å’â€“Ã£Â â€¯ã€ ã„ã€Œè ï¿½ï¼æ€œå–§ç„¡ï¿½ã€ ã ¨è¡¨ç¤ºã •ã,Œã,‹ã ªã ©ã€,

ã€Œæ–‡å—åŒ–ã ‘ã€ ã ¨ã „ã †è¨€è‘‰ã ¯ã€ ã,³ãƒ³ãƒ”ãƒ¥ãƒ¼ã,¿ç’°å¢ƒ §åŽŸå‰‡ ¨ã —ã ¦ãƒžãƒ«ãƒ ãƒ ã,¤ãƒˆæ–‡å—ã,’ä½¿ç”¨ã —ã ªã „æ¬§ç±³ç‰ã ®ãƒ©ãƒ†ãƒ³ã,¢ãƒ«ãƒ•ã,¡ãƒ™ãƒƒãƒˆä½¿ç”¨è ¨€èªžã «ã Šã „ã ¦è©²å½“ã ™ã,‹ç”¨è ªžã Œå¯¾ã ¨ã —ã ªã ‹ã £ã Ÿã “ã ¨ã ‹ã,‰ã€ æ—¥æœ¬èªž ã ®â€œ**Mojibake**â€ ã ¨ã „ã †è¨€è‘‰ã Œã ã ®ã ¾ã ¾é€šç”¨ã ™ã,‹ã,ˆã †ã «ã ªã £ã Ÿã€,â†’#Mojibake

ç›®æ¬¡ [é ˜è¦§ç¤º]

Strange to undecipherable

Swedish example:		Smörgås (open sandwich)
File encoding	**Setting in browser**	**Result**
MS-DOS 437	ISO 8859-1	Sm"rg†s
ISO 8859-1	Mac Roman	Smˆrg Âs
UTF-8	ISO 8859-1	SmÃ¶rgÃ¥s
UTF-8	Mac Roman	Sm√∂rg√•s
UTF-8	ibm/cp037 (EBCDIC)	ë_C¶ÊÅCvË

Character encoding vs character set

- UTF-8 is the character encoding
- Unicode is the character set
- Unicode Consortium curates the character set
- UTF-* stands for Unicode Transform Protocol *

Character encoding vs character set

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

Character encoding vs character set

Character		Binary code point	Binary UTF-8	Hex UTF-8
$	U+0024	010 0100	00100100	24
£	U+00A3	000 1010 0011	11000010 10100011	C2 A3
ह	U+0939	0000 1001 0011 1001	11100000 10100100 10111001	E0 A4 B9
€	U+20AC	0010 0000 1010 1100	11100010 10000010 10101100	E2 82 AC
한	U+D55C	1101 0101 0101 1100	11101101 10010101 10011100	ED 95 9C
𐍈	U+10348	0 0001 0000 0011 0100 1000	11110000 10010000 10001101 10001000	F0 90 8D 88

Emoji modifiers

U+	1F385	1F474	1F44A	1F44D
base				
base + FITZ–1–2				
base + FITZ–3				
base + FITZ–4				
base + FITZ–5				
base + FITZ–6				

JavaScript strings: an Array of code points

	A	א	好	不
Code point	U+0041	U+05D0	U+597D	U+233B4
UTF-8	41	D7 90	E5 A5 BD	F0 A3 8E B4
UTF-16	00 41	05 D0	59 7D	D8 4C DF B4
UTF-32	00 00 00 41	00 00 05 D0	00 00 59 7D	00 02 33 B4

Parts of the stack

- Content-Type header to serve document
- Browser chooses which encoding to use
- JavaScript strings treated correctly
- Content-Type from fetch() to server (defaults to document)
- Back-end server & language
- Connection to database
- Database column type

JSON encoding differences – /🚀/\n

- Browser - "/🚀/\\n"
- Node - "/🚀/\\n"
- Go - "/🚀/\\n"
- Python - "/\ud83d\ude80/\\n"
- PHP - "\/\ud83d\ude80\/\\n" OR "\/🚀\/\\n"
- Java, C++ - depends

Databases

- MySQL 5.6 & AWS Aurora require specifying “utf8mb4”
- Everything else handles UTF-8 by default

String literals: encoding code points

```
const airplane = “\u2708”;
const airplane = “\u{2708}”;
const airplane = “✈”;

const rocket = ”\u{1F680}”;
```

HTML entities code points

- Airplane: ✈
- Rocket: 🚀

- parseInt(“1F680”, 16) => 128640 => 🚀

String methods are problematic

```
> "🚀".length;
< 2
```

```
> "🚀-rocket".charCodeAt(1);
< 56960
```

```
> "🚀-rocket".codePointAt(1);
< 56960
```

```
> "🚀-rocket"[1];
< '\uDE80'
```

Slicing strings

```
> "🚀-rocket".slice(1);
'\uDE80-rocket'
```

```
> [..."🚀-rocket"].slice(1).join("");
'-rocket'
```

```
> Array.from("🚀-rocket").slice(1).join("");
'-rocket'
```

BOM (Byte order mark)

- UTF-8/UTF-16 BOM is \uFEFF
- It has no meaning and should be ignored
- You'll want to strip it

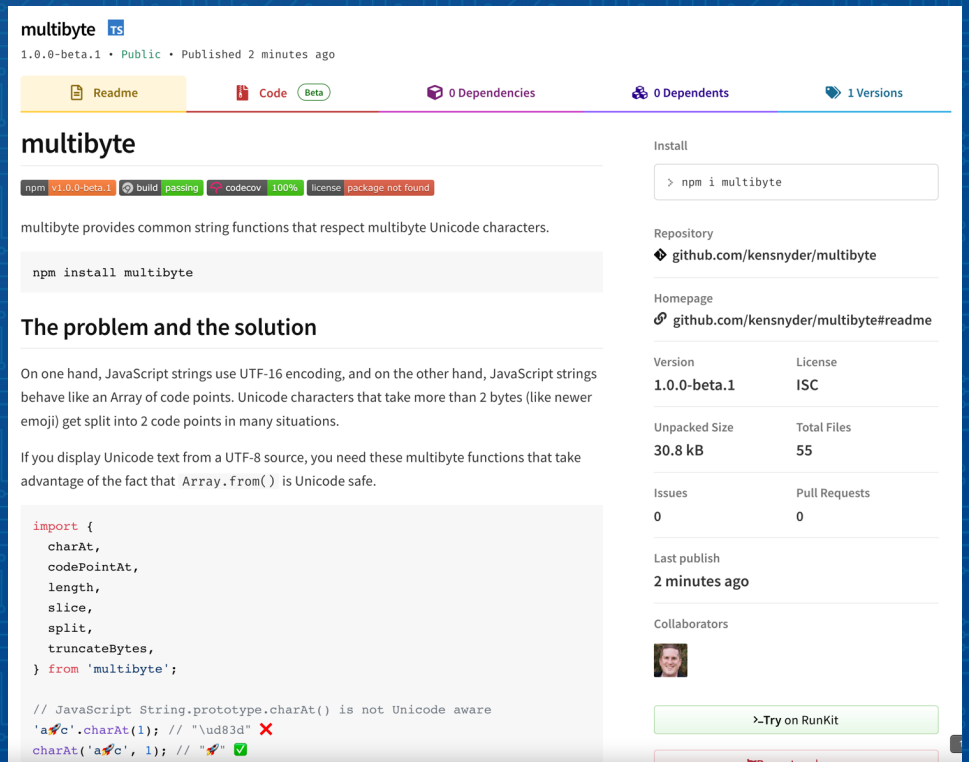
```
> '\uFEFFabc'.length
< 4
> '\uFEFFabc'.slice(0, 1);
< ''
```

Hello npm?


```
function length(str: string) : number {
  return [...str].length;
}

function slice(str: string, from: number = 0, to: number = undefined) : string {
  return [...str].slice(from, to).join('');
}

function codePointAt(str: string, at: number = 0) : number {
  if (at < 0) {
    return undefined;
  }
  for (const char of str) {
    if (at-- === 0) {
      return char.codePointAt(0);
    }
  }
  return undefined;
}

function charAt(str: string, at: number = 0): string {
  if (at < 0) {
    return '';
  }
  for (const char of str) {
    if (at-- === 0) {
      return char;
    }
  }
  return '';
}

```

So I did it

- https://www.npmjs.com/package/multibyte



multibyte TS

1.0.0-beta.1 • Public • Published 2 minutes ago

Readme | Code Beta | 0 Dependencies | 0 Dependents | 1 Versions

multibyte

npm v1.0.0-beta.1 | build passing | codecov 100% | license package not found

multibyte provides common string functions that respect multibyte Unicode characters.

```
npm install multibyte
```

The problem and the solution

On one hand, JavaScript strings use UTF-16 encoding, and on the other hand, JavaScript strings behave like an Array of code points. Unicode characters that take more than 2 bytes (like newer emoji) get split into 2 code points in many situations.

If you display Unicode text from a UTF-8 source, you need these multibyte functions that take advantage of the fact that `Array.from()` is Unicode safe.

```
import {
  charAt,
  codePointAt,
  length,
  slice,
  split,
  truncateBytes,
} from 'multibyte';

// JavaScript String.prototype.charAt() is not Unicode aware
'a🚀c'.charAt(1); // "\ud83d" ❌
charAt('a🚀c', 1); // "🚀" ✅
```

Install

```
> npm i multibyte
```

Repository
github.com/kensnyder/multibyte

Homepage
github.com/kensnyder/multibyte#readme

Version	License
1.0.0-beta.1	ISC

Unpacked Size	Total Files
30.8 kB	55

Issues	Pull Requests
0	0

Last publish
2 minutes ago

Collaborators

>_Try on RunKit

Intl.Collator

JavaScript Demo: Intl.Collator

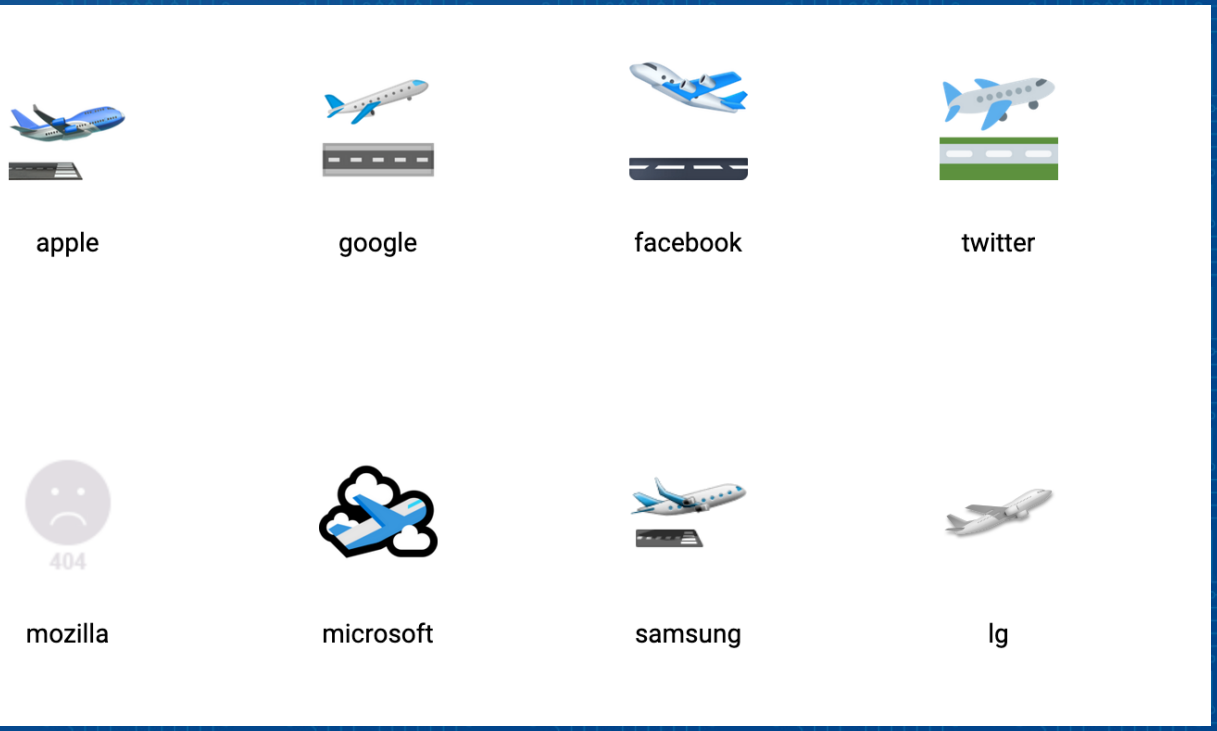
```
1  console.log(['Z', 'a', 'z', 'ä'].sort(new Intl.Collator('de').compare));
2  // Expected output: Array ["a", "ä", "z", "Z"]
3
4  console.log(['Z', 'a', 'z', 'ä'].sort(new Intl.Collator('sv').compare));
5  // Expected output: Array ["a", "z", "Z", "ä"]
6
7  console.log(['Z', 'a', 'z', 'ä'].sort(
8    new Intl.Collator('de', { caseFirst: 'upper' } ).compare
9  ));
10 // Expected output: Array ["a", "ä", "Z", "z"]
11
```

Intl.Segmenter

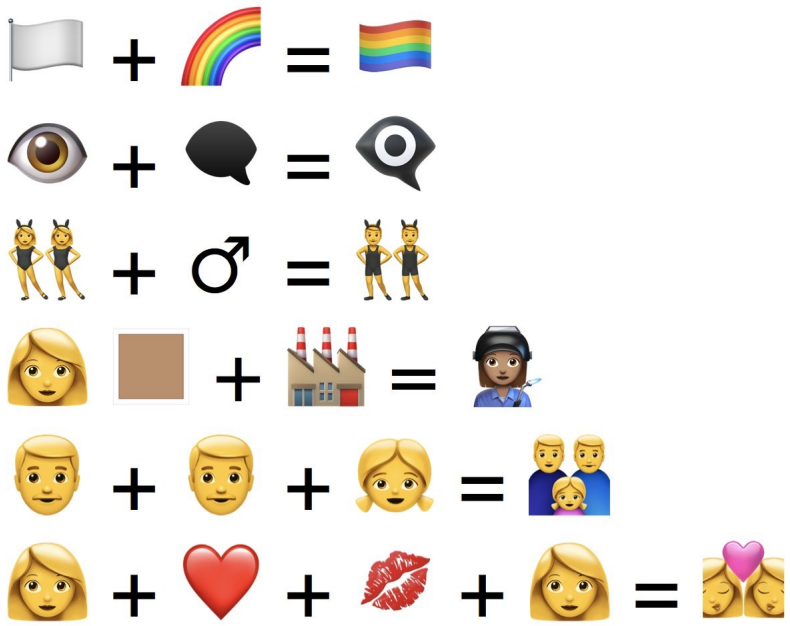
JavaScript Demo: Intl.Segmenter

```
1  const segmenterFr = new Intl.Segmenter('fr', { granularity: 'word' });
2  const string1 = 'Que ma joie demeure';
3
4  const iterator1 = segmenterFr.segment(string1)[Symbol.iterator]();
5
6  console.log(iterator1.next().value.segment);
7  // Expected output: 'Que'
8
9  console.log(iterator1.next().value.segment);
10 // Expected output: ' '
11
```

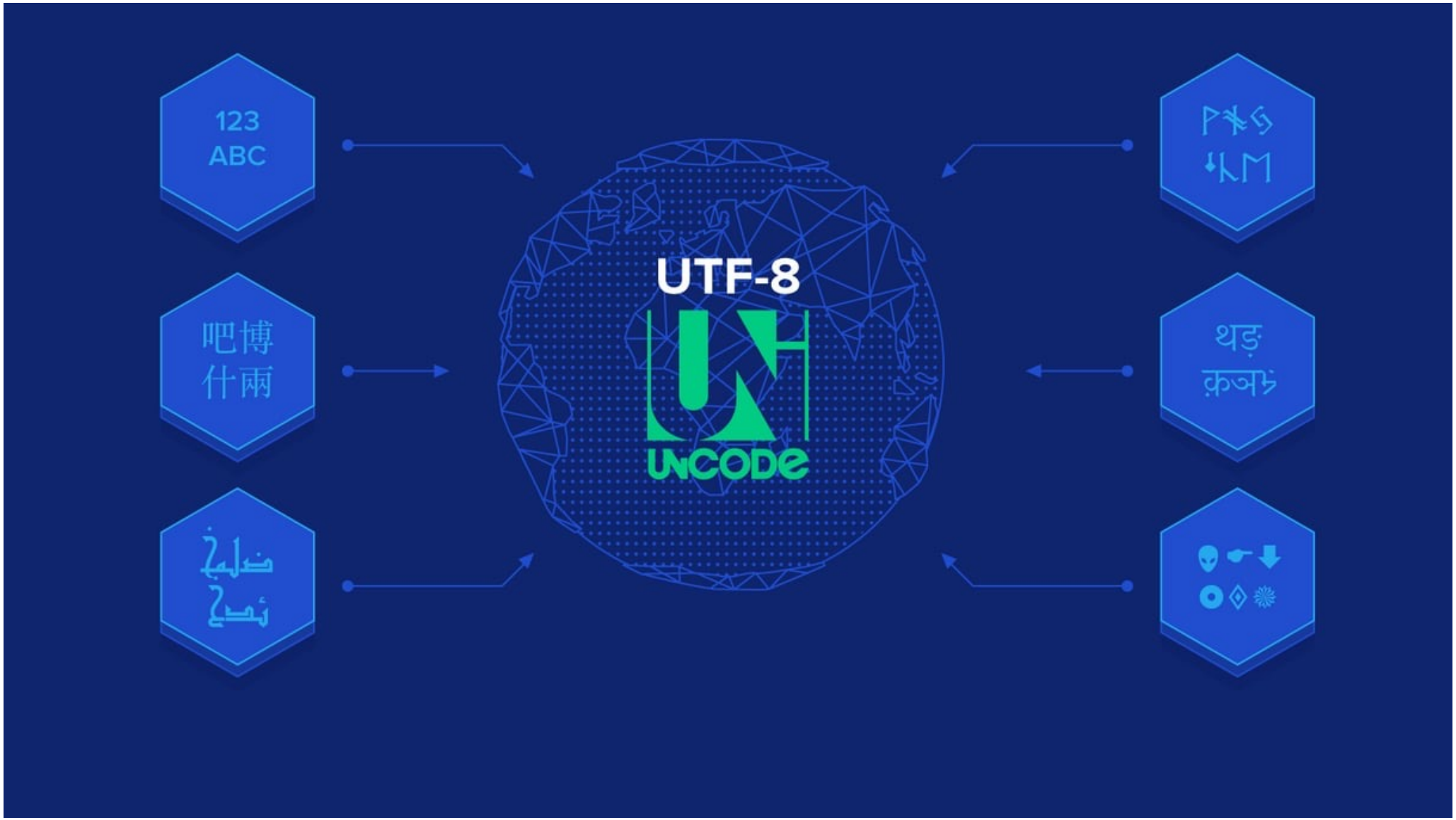
Emoji Implementation



Emoji combinations with ZWJ



Obligatory Zalgo Text


123
ABC
吧博
什兩
UTF-8
UNICODE