

Unicode and its → s: normalisation, Han unification and more

2017

<https://github.com/gyng/book/tree/master/slides/unicode>

Unicode and its ☐糞☐香s: normalisation, Han unification and mʌore

2017 (Shift-JIS edition)

<https://github.com/gyng/book/tree/master/slides/unicode>

1. Some background
2. Unicode and UTF-*x*
3. Programmer pitfalls

```
> 1 + 1;  
← 2
```

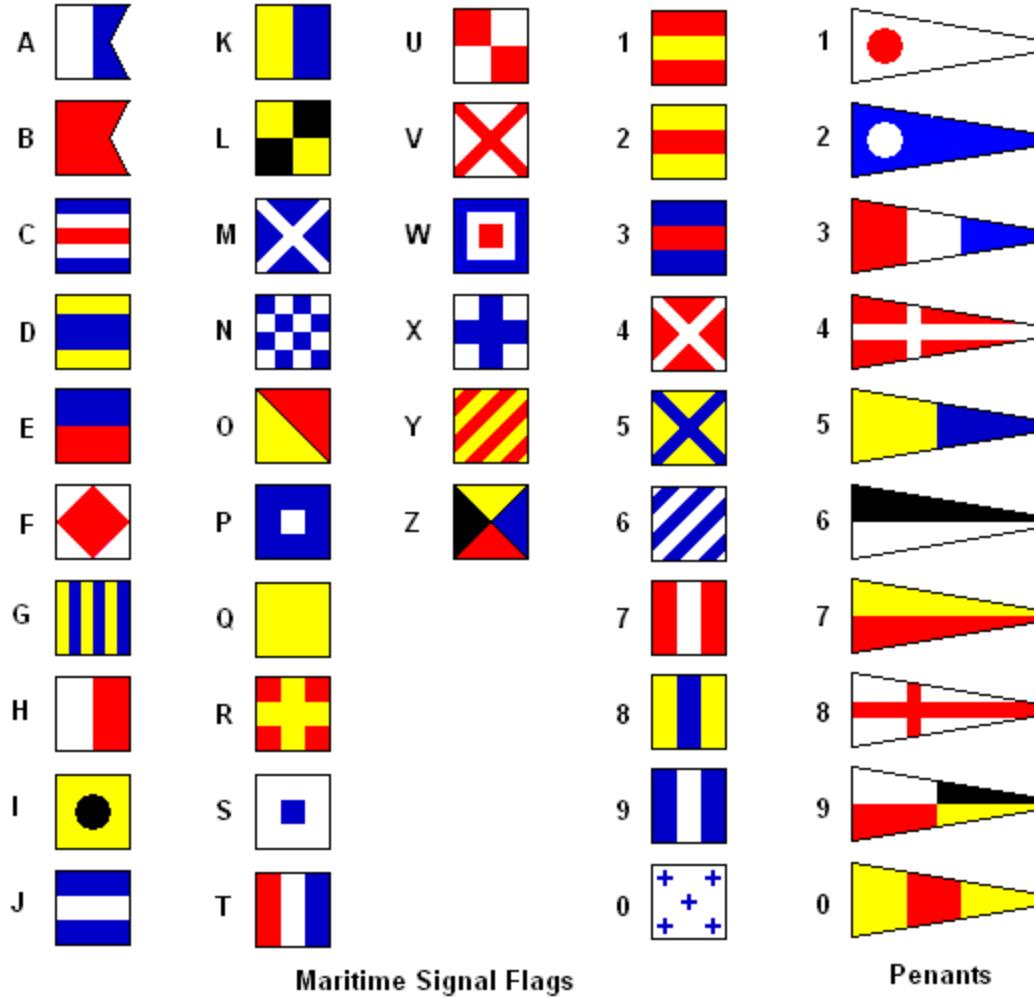
```
> 1 + 1;  
←  SyntaxError: illegal character 
```

Encodings

not encryption

Braille

“Ancient” encodings



“Ancient” encodings — Morse

M	O	R	S	E	C	O	D	E	
---	----	---	...	•	(space)	----	---	---	•

- Three letters: $\{_, ., EOW\}$
- Variable-width letters

○一八三 併	○一六三 低	○一二三 休	○二二三 仰	○一〇三 行	○〇八三 亢	○〇六三 予	○〇四三 夷	○〇二三 中	○〇〇三 七
○一八四 倍	○一六四 住	○一四四 伙	○一二四 仔	○一〇四 仆	○〇八四 交	○〇六四 事	○〇四四 乏	○〇二四 丰	○〇〇四 丈
○一八五 佾	○一六五 佐	○一四五 伯	○一三五 伶	○一〇五 仇	○〇八五 亥	○〇六五 亦	○〇四五 乖	○〇二五 串	○〇〇五 三
○一八六 使	○一六六 佑	○一四六 估	○一二六 仲	○一〇六 今	○〇八六 亦	○〇六六 乘	○〇二六 乘	○〇〇六 上	
○一八七 侃	○一六七 佔	○一四七 佚	○一二七 佚	○一〇七 介	○〇八七 享	○〇六七 二	○〇四七 𠂔	○〇二七 下	○〇〇七 下
○一八八 來	○一六八 何	○一四八 你	○一二八 𠂔	○一〇八 仍	○〇八八 𠂔	○〇六八 于	○〇四八 𠂔	○〇二八 𠂔	○〇〇八 不
○一八九 侈	○一六九 伐	○一四九 伲	○一二九 仵	○一〇九 仔	○〇八九 亨	○〇六九 云	○〇四九 𠂔	○〇二九 丸	○〇〇九 丐
○一九〇 例	○一七〇 余	○一五〇 件	○一三〇 件	○一一〇 仕	○〇九〇 京	○〇七〇 互	○〇五〇 乙	○〇三〇 凡	○〇一〇 丑
○一九一 侍	○一七一 余	○一五一 伶	○一三一 攸	○一一一 他	○〇九一 亭	○〇七一 五	○〇五一 九	○〇三一 丹	○〇一一 且
○一九二 侏	○一七二 佛	○一五二 伸	○一三二 价	○一一二 仗	○〇九二 亮	○〇七二 井	○〇五二 乞	○〇三二 主	○〇一二 丕
○一九三 洫	○一七三 作	○一五三 伺	○一三三 任	○一一三 付	○〇九三 毫	○〇七三 亘	○〇五三 也	○〇三三 世	○〇一三 世
○一九四 侑	○一七四 佞	○一五四 併	○一三四 彷	○一一四 仙	○〇九四 亶	○〇七四 瓦	○〇五四 乩	○〇三四 丘	○〇一四 丘
○一九五 侔	○一七五 佟	○一五五 似	○一三五 企	○一一五 全	○〇九五 亹	○〇七五 况	○〇五五 乳	○〇三五 𠂔	○〇一五 丙
○一九六 侖	○一七六 佩	○一五六 伽	○一三六 伉	○一一六 仞	○〇九六 任	○〇七六 些	○〇五六 乾	○〇三六 父	○〇一六 丞
○一九七 侗	○一七七 侗	○一五七 佃	○一三七 伊	○一一七 任	○〇九七 西	○〇七七 剗	○〇五七 剗	○〇三七 乃	○〇一七 王

一

、

J

乙

J

二

上

“Ancient” encodings — Chinese telegraph code

電 碼

7193 4316

--... .--- -.-. .--- / -.... .--- -....

EGL EWS

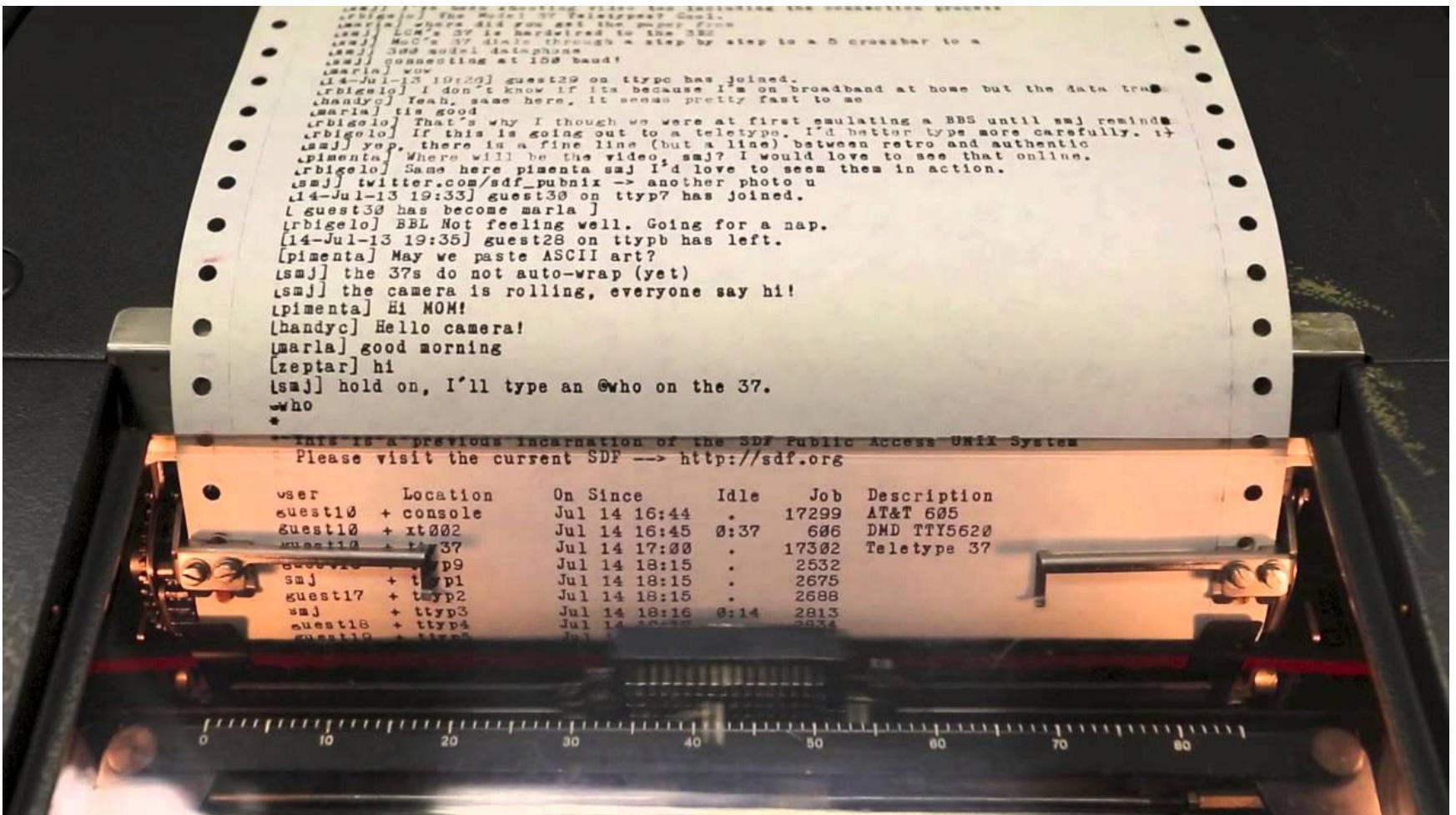
. ---. .-.. / . .-- ...

- Same characters, different encodings, different lengths
- The *code point* (電 = 7193) is not the *encoding* (morse)

ASCII

00	NUL	20		40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	\$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(48	H	68	h
09	HT	29)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
:	:	:	:		:		:

<http://www.catb.org/esr/faqs/things-every-hacker-once-knew/>



<https://youtu.be/MikoF6KZjm0?t=289>

ASCII

- 1963
- American Standard Code for Information Interchange
- 7-bit (8^{th} bit used for parity)
- $2^7 = 128$ possible values
- Has a method to its madness!

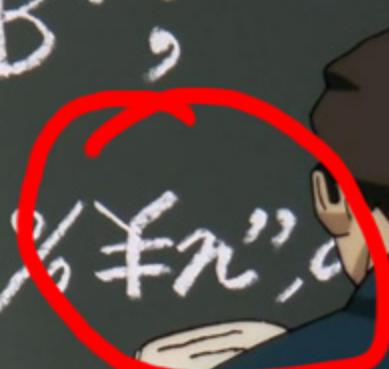
ASCII

- 0–31 are control characters NUL CR LF DEL
- 32–126 are punctuation, numerals and letters
- `_` in binary: `0100000` = 32 = 0x20
- `A` in binary: `1000001` = 65 = 0x41
- `a` in binary:
 - = $65 + 32$
 - = $0x41 + 0x20$
 - = `1000001 | 0100000`

Modified ASCII

- Extended ASCII (8-bit, has more characters ç ü ☒ ¶ æ)
- Modified 7-bit ASCII exist
 - # → £ on UK teletypes
 - \ → ¥ in Japan (Shift-JIS)
 - \ →₩ in Korea (EUC-KR)

```
a='A'; b='B';  
printf("%c>%c\n",a,b);  
printf("%c=%c",a,a);  
printf("%c<%c  
=123;  
~~~~~
```



ASCII Highlights

Control characters

- CR Moves the print head to the left margin
- LF Scrolls down one line
- DEL Backspace and delete
- ETX ^C (SIGINT)
- EOT ^D
- BEL Rings the (physical) bell

```
sleep 3 && echo $'\a'
```

ASCII ⇔ Unix/Linux *control codes*

Hex	Char	Hex	Char
00	NUL '\0' (null character)	40	@
01	SOH (start of heading)	41	A
02	STX (start of text)	42	B
03	ETX (end of text)	43	C ➤
04	EOT (end of transmission)	44	D ➤
05	ENQ (enquiry)	45	E
06	ACK (acknowledge)	46	F
07	BEL '\a' (bell)	47	G
08	BS '\b' (backspace)	48	H ➤
09	HT '\t' (horizontal tab)	49	I
:			

man ascii

So, what's the problem with ASCII?

ASCII
^

Problems with ASCII

- Latin-centric
- Everybody else came up with their own encodings
- Alternative ASCII sets cause problems with interchange
 - moji bake
- Mojibake (文字化け): JIS, Shift-JIS, EUC, and Unicode
- No emoji, only emoticons :-(

Dark ages

- ???
- ???
- ???
- ???
- ???
- Xerox Character Code Standard (XCCS), 1980

Early telecommunications	ASCII · ISO/IEC 646 · ISO/IEC 6937 · T.61 · BCDIC · Baudot code · Morse code (telegraph code Wabun code) · Special telephony codes: Non-Latin, Chinese, Cyrillic · Needle telegraph codes
ISO/IEC 8859	-1 · -2 · -3 · -4 · -5 · -6 · -7 · -8 · -9 · -10 · -11 · -12 · -13 · -14 · -15 · -16
Bibliographic use	ANSEL · ISO 5426 / 5426-2 / 5427 / 5428 / 6438 / 6861 / 6862 / 10585 / 10586 / 10754 / 11822 · MARC-8
National standards	ArmSCII · BraSCII · CNS 11643 · ELOT 927 · GOST 10859 · GB 18030 · HKSCS · ISCII · JIS X 0201 · JIS X 0208 · JIS X 0212 · JIS X 0213 · KOI-7 · KPS 9566 · KS X 1001 · PASCII · SI 960 · TIS-620 · TSCII · VISCII · YUSCII
EUC	CN · JP · KR · TW
ISO/IEC 2022	CN · JP · KR · CCCII
MacOS code pages ("scripts")	Arabic · Celtic · CentEuro · ChineseSimp / EUC-CN · ChineseTrad / Big5 · Croatian · Cyrillic · Devanagari · Dingbats · Esperanto · Farsi · Gaelic · Greek · Gujarati · Gurmukhi · Hebrew · Iceland · Japanese / ShiftJIS · Korean / EUC-KR · Latin-1 · Roman · Romanian · Sámi · Symbol · Thai / TIS-620 · Turkish · Ukrainian
DOS code pages	100 · 111 · 112 · 113 · 151 · 152 · 161 · 162 · 163 · 164 · 165 · 166 · 210 · 220 · 301 · 437 · 449 · 489 · 620 · 667 · 668 · 707 · 708 · 709 · 710 · 711 · 714 · 715 · 720 · 721 · 737 · 768 · 770 · 771 · 772 · 773 · 774 · 775 · 776 · 777 · 778 · 790 · 850 · 851 · 852 · 853 · 854 · 855/872 · 856 · 857 · 858 · 859 · 860 · 861 · 862 · 863 · 864/17248 · 865 · 866/808 · 867 · 868 · 869 · 874/1161/1162 · 876 · 877 · 878 · 881 · 882 · 883 · 884 · 885 · 891 · 895 · 896 · 897 · 898 · 899 · 900 · 903 · 904 · 906 · 907 · 909 · 910 · 911 · 926 · 927 · 928 · 929 · 932 · 934 · 936 · 938 · 941 · 942 · 943 · 944 · 946 · 947 · 948 · 949 · 950/1370 · 951 · 966 · 991 · 1034 · 1039 · 1040 · 1041 · 1042 · 1043 · 1044 · 1046 · 1086 · 1088 · 1092 · 1093 · 1098 · 1108 · 1109 · 1114 · 1115 · 1116 · 1117 · 1118 · 1119 · 1125/848 · 1126 · 1127 · 1131/849 · 1139 · 1167 · 1168 · 1300 · 1351 · 1361 · 1362 · 1363 · 1372 · 1373 · 1374 · 1375 · 1380 · 1381 · 1385 · 1386 · 1391 · 1392 · 1393 · 1394 · Kamenický · Mazovia · CWI-2 · KOI8 · MIK · Iran System
IBM AIX code pages	367 · 371 · 806 · 813 · 819 · 895 · 896 · 912 · 913 · 914 · 915 · 916 · 919 · 920 · 921/901 · 922/902 · 923 · 952 · 953 · 954 · 955 · 956 · 957 · 958 · 959 · 960 · 961 · 963 · 964 · 965 · 970 · 971 · 1004 · 1006 · 1008 · 1009 · 1010 · 1011 · 1012 · 1013 · 1014 · 1015 · 1016 · 1017 · 1018 · 1019 · 1029 · 1036 · 1089 · 1111 · 1124 · 1129/1163 · 1133 · 1350 · 1382 · 1383
IBM Apple Macintosh Emulations	1275 · 1280 · 1281 · 1282 · 1283 · 1284 · 1285 · 1286
IBM Adobe Emulations	1038 · 1276 · 1277
IBM DEC Emulations	1020 · 1021 · 1023 · 1090 · 1100 · 1101 · 1102 · 1103 · 1104 · 1105 · 1106 · 1107 · 1287 · 1288
IBM HP Emulations	1050 · 1051 · 1052 · 1053 · 1054 · 1055 · 1056 · 1057 · 1058
Windows code pages	CER-GS · 874/1162 (TIS-620) · 932/943 (Shift JIS) · 936/1386 (GBK) · 950/1370 (Big5) · 949/1363 (EUC-KR) · 1169 · 1174 · Extended Latin-8 · 1200 (UTF-16LE) · 1201 (UTF-16BE) · 1250 · 1251 · 1252 · 1253 · 1254 · 1255 · 1256 · 1257 · 1258 · 1259 · 1261 · 1270 · 54936 (GB18030)
EBCDIC code pages	1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11 · 12 · 13 · 14 · 15 · 16 · 17 · 18 · 19 · 20 · 21 · 22 · 23 · 24 · 25 · 26 · 27 · 28 · 29 · 30 · 31 · 32 · 33 · 34 · 35 · 36 · 37/1140 · 38 · 39 · 40 · 251 · 252 · 254 · 256 · 257 · 258 · 259 · 260 · 264 · 273/1141 · 274 · 275 · 276 · 277/1142 · 278/1143 · 279 · 280/1144 · 281 · 282 · 283 · 284/1145 · 285/1146 · 286 · 287 · 288 · 289 · 290 · 293 · 297/1147 · 298 · 300 · 310 · 320 · 321 · 322 · 330 · 351 · 352 · 353 · 355 · 357 · 358 · 359 · 360 · 361 · 363 · 382 · 383 · 384 · 385 · 386 · 387 · 388 · 389 · 390 · 391 · 392 · 393 · 394 · 395 · 410 · 420/16804 · 421 · 423 · 424/8616/12712 · 425 · 435 · 500/1148 · 803 · 829 · 833 · 834 · 835 · 836 · 837 · 838/838 · 839 · 870/1110/1153 · 871/1149 · 875/4971/9067 · 880 · 881 · 882 · 883 · 884 · 885 · 886 · 887 · 888 · 889 · 890 · 892 · 893 · 905 · 918 · 924 · 930/1390 · 931 · 933/1364 · 935/1388 · 937/1371 · 939/1399 · 1001 · 1002 · 1003 · 1005 · 1007 · 1024 · 1025/1154 · 1026/1155 · 1027 · 1028 · 1030 · 1031 · 1032 · 1033 · 1037 · 1047 · 1068 · 1069 · 1070 · 1071 · 1073 · 1074 · 1075 · 1076 · 1077 · 1078 · 1079 · 1080 · 1081 · 1082 · 1083 · 1084 · 1085 · 1087 · 1091 · 1097 · 1112/1156 · 1113 · 1122/1157 · 1123/1158 · 1130/1164 · 1132 · 1136 · 1137 · 1150 · 1151 · 1152 · 1159 · 1165 · 1166 · 1278 · 1279 · 1303 · 1364 · 1376 · 1377 · JEF · KEIS
Platform specific	Acorn · Adobe Standard · ATASCII · Atari ST · BICS · Casio calculators · CDC · CPC · DEC Radix-50 · DEC MCS/NRCS · DG International · ELWRO-Junior · FIELDATA · GEM · GEOS · GSM 03.38 · HP Roman Extension · HP Roman-8 · HP Roman-9 · HP calculators · LICS · LMBCS · NEC APC · NeXT · PETSCII · Sharp calculators · TI calculators · Ventura International · Ventura Symbol · WISCII · XCCS · ZX80 · ZX81 · ZX Spectrum
Unicode / ISO/IEC 10646	UTF-1 · UTF-7 · UTF-8 · UTF-16 (UTF-16LE/UTF-16BE) / UCS-2 · UTF-32 (UTF-32LE/UTF-32BE) / UCS-4 · UTF-EBCDIC · GB 18030 · BOCU-1 · CESU-8 · SCSU
Miscellaneous code pages	ABICOMP · APL · Cork · HZ · Johab · SEASCII · TACE16 · TRON · UTF-5 · UTF-6 · WTF-8
Related topics	Code page · Control character (C0 C1) · CCSID · Character encodings in HTML · Charset detection · Han unification · Hardware · ISO 6429/IEC 6429/ANSI X3.64 · Mojibake

Unicode

Timeline of Unicode

- 1985, Sapporo, 🏛️
- KanjiTalk, localised 🎨
- Shift-JIS is a 💩
- Bunch of 🧑 start working on Unicode specs
- 1988, submitted to ISO 📄
- 1991, Han Unification accepted 😊
- 1992, 🙌 *Kiss Your ASCII Goodbye in PC Magazine*
- 1995, ☕ Java 1.0 launches with Unicode support

<http://www.unicode.org/history/earlyyears.html>



The first Unicode TV interview (1991)

<http://www.unicode.org/history/unicodeMOV.mov>

In that video, the VP of Unicode made:

- three statements
- three inaccuracies (in 2017)

Julie Allen

Editor



Unicode: the Movie (2000)

<http://www.unicode.org/history/movie/UniMovie-large.mov>

Unicode features*

- A common representation for all characters
- \simeq Compatible with ASCII for English ($A = 65$)
- Efficient encoding
- ~~Uniform width encoding~~
- Han unification (CJK languages share glyphs)

Unicode 10.0 (2017 June 20)

Unicode 10.0 adds 8,518 characters, for a total of 136,690 characters

<http://www.unicode.org/versions/Unicode10.0.0/>

56 emoji (2,666 total)

http://www.unicode.org/reports/tr51/tr51-12.html#Emoji_Counts

Bitcoin sign

...and more

Unicode terminology

- Scalar value € U+20AC EURO SIGN
- Range U+0000..U+FFFF
- Sequence É <U+0045 LATIN CAPITAL LETTER E, U+0301 COMBINING ACUTE ACCENT>

Unicode planes

- `U+0000..U+FFFF` is Plane 0, Basic Multilingual Plane (BMP)
- Each plane encodes up to $2^{16} = 65536$ code points
- Commonly used characters

Standard

Unicode

Encoding

UTF-8, UTF-16, UTF-32, UCS-2, UCS-4

UTF-16

- Early UTF-16 was fixed-width (UCS-2)
- 2 or 4 bytes per character
- 2 bytes for characters in BMP
 - Can be more efficient than UTF-8 for CJK (2B vs 3B)
- Surrogate pairs have to be handled for code points outside BMP
 - Byte-order matters

UTF-32

- 32 bits ought to be enough for anybody

UTF-32

- A now takes up 4 bytes

SCSU

But wait! There's more!

⌘ Standard Compression Scheme for Unicode ⌘

<http://www.unicode.org/reports/tr6/>

SCSU

♪リンゴ可愛いや可愛いやリンゴ。半世紀も前に流行した「リンゴの歌」がぴったりするかもしれない。米アップルコンピュータ社のパソコン「マック（マックintosh）」を、こよなく愛する人たちのことだ。「アップル信者」なんて言い方まである。

= not compressible 18/12 = 1.5

= 3000 - 307F static window 7 12/11 = 1.1

= 3040 - 309F dynamic window 5 45/14 = 4.2

= 30A0 - 30FF dynamic window 6 38/8 = 4.75

= FF00 - FF7F dynamic window 7 2/2 = 1.00

= 2600-267F 1/1 = 1.00

- Do not use it*

UTF-8

- Variable width
- Single-byte (Same as ASCII, 7-bits)

00100100
└ Is single-byte

= 36 = 0x24 = \$ U+0024 DOLLAR SIGN

UTF-8

- Multi-byte

```
1110aaaa 10bbbbbb 10cccccc
|         |         |
|         |         |
|         |         Is continuation byte
|         |
|         2 continuation bytes
|         |
|         Is multi-byte
```

- First byte specifies number of continuation bytes
- Encoded character is aaaabbbb bbcccccc

Private use areas

- U+E000..U+F8FF , U+F0000..U+FFFFD , U+100000..U+10FFFF
- Suggested for internal use
 - data processing
 - artificial scripts
 - ancient scripts
- ☐ U+F8FF (↑ - ↵ - k)
- Ubuntu has U+E0FF and U+F200

U+E0FF: ☐

U+F200: ubuntu®

Combining characters

- Modify other characters

e + ' = é

```
<e U+0065 LATIN SMALL LETTER E,  
`U+0301 COMBINING ACUTE ACCENT>
```

- Precomposed é

```
é U+00E9 LATIN SMALL LETTER E WITH ACUTE
```

- Modifiers come after base character

Unicode normalisation

- Some combined characters are sort of the same
- Equivalence criteria
 - canonical (NF)
 - compatibility (NFK)
- `ff` U+FB03 LATIN SMALL LIGATURE FFI vs `f` `f` `í`
 - not equivalent under canonical (NF)
 - equivalent under NFK compatibility (NFK)
- NF is used to canonicalise combining characters

Unicode normalisation

- *NFD Normalization Form Canonical Decomposition*
- *NFC Normalization Form Canonical Composition*
- *NFKD Normalization Form Compatibility Decomposition*
- *NFKC Normalization Form Compatibility Composition*

Han unification

- Maps common Chinese, Japanese, Korean (CJK) characters into unified set

UNICODE U+66DC LANGUAGE Traditional Chinese	UNICODE U+66DC LANGUAGE Simplified Chinese	UNICODE U+66DC LANGUAGE Japanese	UNICODE U+66DC LANGUAGE Korean
			

- Different countries have different standards

Han unification

- Variants can be significant (names)

ashi

芦 Ashi·da, given name vs Ashi·ya, old place name

芦田さんは芦屋のお嬢様だ

- Educational software
- People get 😠 over the differences

Han unification

CJK Extension F contains mostly rare characters, but also includes a number of personal and placename characters important for government specifications in Japan, in particular.

CJK Extension F was added in Unicode 10.0 (2017)

Han unification

- Lose round-trip conversion compatibility with character sets which have variants

<https://support.microsoft.com/en-us/help/170559/prb-conversion-problem-between-shift-jis-and-unicode>

Han unification

- Can use Unicode variation selectors

U+E0101 VARIATION-SELECTOR-18

» "刃\ufe04"

← "刃"

» "刃\uDB40\uDD01"

← "刃"

<http://www.unicode.org/ivd/>

<http://unicode.org/reports/tr37/>

Control sequences and vertical text

- Vertical text
- RTL mark

The screenshot shows a section of an Arabic Wikipedia article. At the top, there's a navigation bar with links for 'غير مسجل للدخول' (Not logged in), 'نقاش مساهمات' (Talk), 'إنشاء حساب' (Create account), 'دخول' (Log in), 'أبحث في ويكيبيديا' (Search Wikipedia), 'عدل' (Edit), 'ال التاريخ' (History), 'اقرأ' (Read), and 'نقاش' (Talk). Below the navigation bar, there's a sidebar with links for 'الحسابات الاجتماعية الرسمية' (Official social media accounts) and 'لوبيكبيديا العربية' (Arabic Wikipedia logo). There are also social media links for 'فيسبوك' (Facebook), 'تويتر' (Twitter), 'إنستغرام' (Instagram), and 'فيس بوك' (Facebook). The main content area has a heading 'قائمة الحروف العربية المشتقة' (List of Arabic-derived characters) with an edit link. The text discusses the formation of these characters from the Arabic alphabet and lists three categories: 'الشكل' (Shape), 'بالهمز أو النقط وما إليها' (With hamza or dots and what it includes), and 'ربط الحروف ودمجها' (Linking characters and combining them). A sidebar on the right contains links for 'تصفح' (Browse), 'المواضيع' (Topics), 'أبجدي' (Alphabetic), 'بوابات' (Portals), and 'مقالة عشوائية' (Random article).

Unicode Bidirectional Algorithm @ <http://unicode.org/reports/tr9/>
Unicode Vertical Text Layout @
<http://www.unicode.org/reports/tr50/>

Ligatures

Unicode maintains that ligaturing is a presentation issue rather than a character definition issue

- But! There are some predefined ligatures

ff U+FB04 LATIN SMALL LIGATURE FFL

Av U+A738 LATIN CAPITAL LETTER AV

æ U+00E6 LATIN SMALL LETTER AE

- Similar issue with subscript and superscript

Emoji

- 絵 (\cong picture) + 文字 (\cong written character)
- Early emoji were created by Japanese telcos
- 2008: Gmail, iPhone
- 2010: Unicode 6
- 禁 空 合 満 有 月 申 割 営 NG OK 可 ツ ツ サ シ ハ

<http://unicode.org/reports/tr51/>

Can be represented differently



- This is a problem



iOS



Android



Windows



Samsung



LG



HTC



Facebook



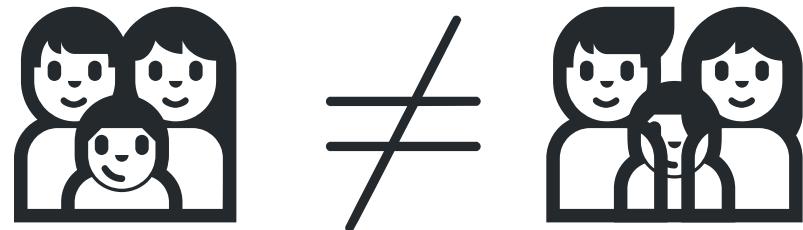
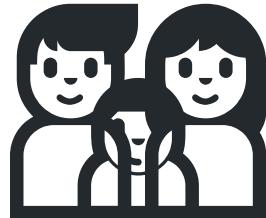
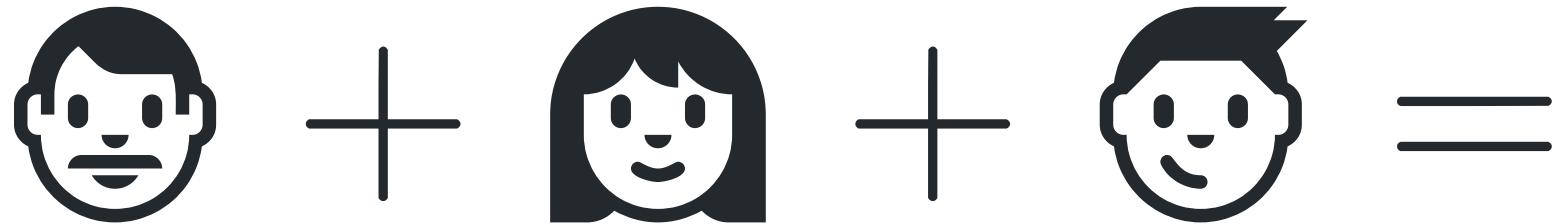
Twitter



	Apple	Google	Microsoft	Samsung	Facebook	Twitter
2013						
2014						
2015						
2016						
2017						
2018						

<https://blog.emojipedia.org/google-updates-gun-emoji/>

Combining emoji

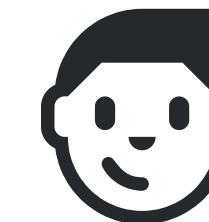
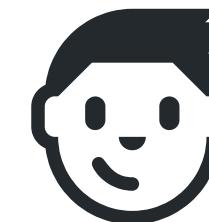
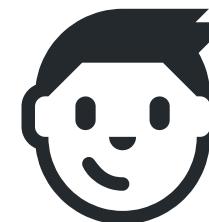
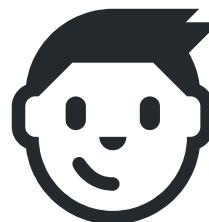
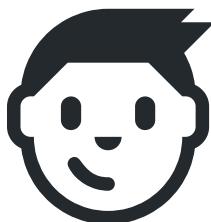
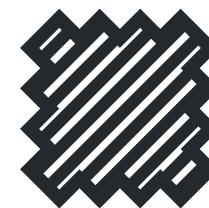
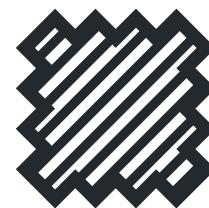
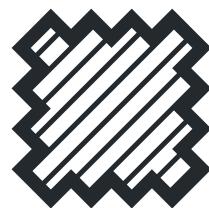
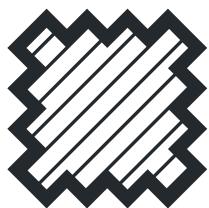
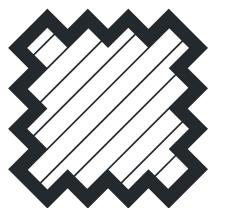


U+1F46A FAMILY vs combined character

$$S + G = SG$$
$$G + S = GS$$

```
S < U+1F1F8 REGIONAL INDICATOR SYMBOL LETTER S >
G < U+1F1EC REGIONAL INDICATOR SYMBOL LETTER G >
```

Variation selectors



<http://unicode.org/faq/vs.html>



EarthWeb commercial, 2001

<http://www.unicode.org/history/EarthwebCommercial.avi>



Necessary
but not necessarily sufficient
programmer knowledge



Recognise garbled text as mojibake

- Maybe able to recover content by swapping character sets
- UTF-8 seen using KOI8-R, a Cyrillic character set

п я—п ||п ъп ъп ъя ||п ъя—я ■

UTF-8

Библиотека

Use UTF-8 for all source code

- Configure your text editor
- Magic comments for some languages

Ruby \leq 1.9.x

```
# encoding: UTF-8
```

² Python 2

```
# -*- coding: utf-8 -*-
```

C \leq C99

```
/* Dear future programmer: Good Luck  */
```

Text processing

- Treat input as bytes
- Treat text as strings (and not byte arrays)
- Use UTF-8 wherever possible
 - unless you know what you are doing
- Decide what to do with invalid bytes
 - discard or substitute?
- Do not self-roll your own text encoding library

Read in text with the right encoding

Especially when parsing HTML or XML

```
# Nokogiri
doc = Nokogiri.XML(html, nil, 'EUC-JP')
```

```
# Beautiful Soup
soup = BeautifulSoup(html, fromEncoding='Shift_JIS')
```

Set HTML charset

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
</html>
```

Use `lang` in HTML as needed

```
<html lang="en">
  <body>
    <span lang="zh-Hans">刃</span>
    <span lang="zh-Hant">刃</span>
    <span lang="ja">刃</span>
    <span lang="ko">刃</span>
    <span lang="vi">刃</span>
  </body>
</html>
```

U+5203	刃	刃	刃	刃	刃	knife edge

Case conversion

- What is the uppercase form of `i` ?

Case conversion

- What is the uppercase form of i ? I
- In Turkish?

Case conversion

- What is the uppercase form of `i` ?
- In Turkish?

`i` → `I`

`i` → `İ`

Case conversion

- What is the uppercase form of `i` ?
- In Turkish?

`I` → `I`

`i` → `İ`

- In Turkish/English mixed text?

Case conversion

- Harder than you think
- What is the uppercase form of

ß U+00DF LATIN SMALL LETTER SHARP S ?

Case conversion

- DE German
- ß upcases to ss

Case conversion

- `ß` upcases to `SS`
- ...or `U+1E9E ß LATIN CAPITAL LETTER SHARP S`

http://unicode.org/faq/casemap_charprop.html

Case conversion

In 2016, the Council for German Orthography proposed the introduction of optional use of ß in its ruleset (i.e. variants STRASSE vs. STRAßE would be accepted as equally valid).[9] The rule was officially adopted in 2017.[10]

Does your favourite programming language work?

JavaScript (Firefox 53)

```
>> 'ß'.toLocaleUpperCase('de-DE');
'ß'
```

JavaScript (Chrome 59)

```
>> 'ß'.toLocaleUpperCase('de-DE');
'ss'
```

Does your favourite programming language work?

⌚² Python 2

```
>>> u'ß'.upper()  
u'\xdf' # β
```

⌚³ Python 3

```
>>> 'ß'.upper()  
'SS'
```

Does your favourite programming language work?

💎 Ruby 2.3

```
> "\u{00df}".upcase  
=> "ß"
```

💎 Ruby 2.4

```
> "\u{00df}".upcase  
=> "SS"
```

Does your favourite programming language work?



```
public class UppercaseThis {  
    public static void main(String[] args) {  
        System.out.println("\u00df".toUpperCase());  
    }  
}
```

SS



```
fn main() { println!("{}", "ß".to_uppercase()); }
```

SS

Use variation selectors as needed

U+E0101 VARIATION-SELECTOR-18

» "刃\ufe04"

← "刃"

» "刃\uDB40\uDD01"

← "刃"

Use a correct font for the language outside HTML

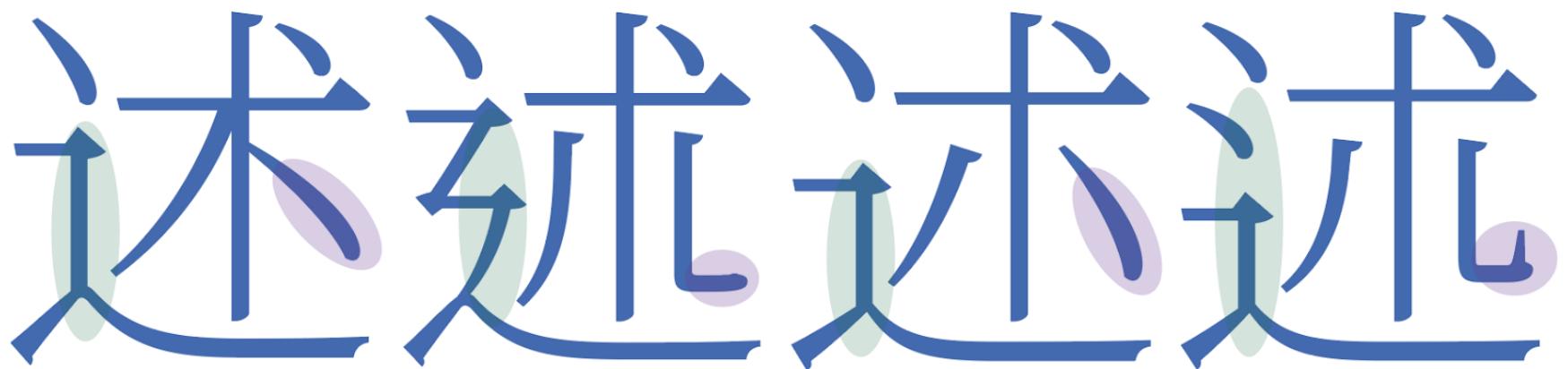
- Google's Noto/Noto CJK has great support
- Similarly, Adobe's Source Han

<https://www.google.com/get/noto/help/cjk/>

<https://source.typekit.com/source-han-serif>

Use a correct font for the language outside HTML

Glyph variations



述 U+8FF0 in S. Chinese, T. Chinese, Japanese and Korean

Noto Serif CJK

Use a correct font for the language outside HTML

Vertical text support

セイリツシユ語族は太平洋岸北西部(カナダのブリティッシュコロニア州、およびアメリカ合衆国のワシントン州、オレゴン州、アイダホ州、モンタナ州)で用いられている言語群である。セイリッシュ語族の分類に関しては、まず Boas & Haeberlin (1927) で 20 種類の言語が「方言」として内陸語派

セイリツシユ語族は太平洋岸北西部(カナダのブリティッシュコロニア州、およびアメリカ合衆国のワシントン州、オレゴン州、アイダホ州、モンタナ州)で用いられている言語群である。セイリッシュ語族の分類に関しては、まず Boas & Haeberlin (1927) で 20 種類の言語が「方言」として内陸語派

Noto Serif CJK

<https://helpx.adobe.com/photoshop/user-guide.html?topic=/photoshop/morehelp/text.ug.js>



Unencoded characters

How can I display (CJK/my own) characters not encoded in Unicode?



UTC-00791



UTC-01312

biáng, from *biángbiáng* 面, a noodle dish from Shaanxi, China

Coming to a Unicode version soon?

Unencoded characters

- Use an image
- Use Ideographic Description Sequences

□日□□□□□□□□□□□ U+2FF0..U+2FFF

書史 for 鼾

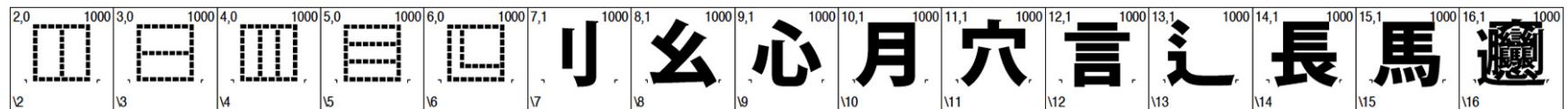
- Use fonts which have the unencoded glyph either
 - as an existing character (Wingdings ! ☺ ☻ ☻ ☻)
 - in Private Use Area
 - as a combined sequence

Unencoded characters

- Source Han and Noto have glyphs for *biáng!*
- Uses Unicode and font features to combine existing glyphs
 - Ideographic Description Characters
 - OpenType's `ccmp` (Glyph Composition/Decomposition)
 - Ligatures `liga`

<https://blogs.adobe.com/CCJKT/2014/03/ids-opentype.html>

Unencoded characters



□辶目穴月□□□□彡長曰言馬曰彡長刂心 (*traditional*)
□辶目穴月□□□□彡长曰言马曰彡长刂心 (*simplified*)

<https://blogs.adobe.com/CCJKType/2017/04/designing-implementing-biang.html>



What 長 馬 長 月 么 言 長 馬 長 小心 長 馬 長 月 么 言
么 長 馬 長 小心 面 looks like

String sorting

- Sorting strings is hard!

```
>> 'é' > 'f'  
true
```

String sorting

- A-ha! Can we use normalisation for this?

```
>> 'café'.normalize('NFKD')  
'cafe '
```

String sorting

- Sometimes

```
>> '한국어'.normalize('NFKD')
"한국어"
```

spaces manually added

MDN: [String.prototype.normalize\(\)](#)

String sorting and equality

- Use a locale-aware comparison

```
>> ['Aa', 'Äa', 'Äb', 'Ab'].sort();
['Aa', 'Ab', 'Äa', 'Äb']
```

```
>> ['Aa', 'Äa', 'Äb', 'Ab']
>> .sort(a, b => a.localeCompare(b, 'de'));
['Aa', 'Äa', 'Ab', 'Äb']
```

[MDN: String.prototype.localeCompare\(\)](#)

String searching

- How do I search for `café` by typing `cafe`, or `cafe``?

String searching

- Not easy!
- Locale-aware comparisons
- Unicode-aware regex

String searching (proper)

- Read *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard* by Richard Gillam
- Read <http://unicode.org/reports/tr10/#Searching>

Asymmetric searching

query	matches
resume	resume, Resume, RESUME, résumé, rèsomè, Résumé, ...
résumé	résumé, Résumé, RÉSUMÉ, ...
けんこ	けんこ, ケンコ, げんこ, けんご, ゲンコ, ケンゴ, ...

String length

Problems arise when your string contains

- combining marks
- surrogate pairs (UTF-16)

String length — combined characters

What's the length of `café` ?

String length — combined characters

```
>> 'café'.length  
5  
  
>> 'café'.normalize().length  
4
```

```
>> ' UNICODE '.length  
5  
  
>> ' UNICODE \u3099 '.normalize().length  
5
```

Should generally work for combined characters 

String length — surrogate pairs

What's the length of  U+1F4A9 PILE OF POO ?

- UTF-8
F0 9F 92 A9
- Surrogate pairs (UTF-16)
D83D DCA9

Does your favourite programming language work?

💩 JavaScript

```
>> '💩'.length  
2  
>> [...'💩'].length  
1
```

💩² Python 2

```
>>> len(u'💩')  
2
```

💩³ Python 3

```
>>> len('💩')  
1
```

Does your favourite programming language work?

💎 Ruby

```
>> '💩'.length  
1
```

☕ Java

```
System.out.println("💩".length());  
// 2  
  
// This margin is too small to contain the solution  
// Use java.text.BreakIterator
```

⚙️ Rust

```
println!("{}", "💩".len());  
// 4  
  
println!("{}", "💩".chars().count());  
// 1
```

Regex

- What if you want to match `e` and `é`?
- What about all the different whitespace characters?
- What if I want to match one character `/^.$/` but my character is combined? `é` \neq `e` + `'`
- What about matching non-Latin characters?

Regex

- Use Regex right
- Make sure `\w` `\d` `\s` are Unicode-aware
- Make sure your Regex engine does **case-folding**
- Match by Unicode (Perl)
 - `\N{}` Named or numbered (Unicode) char or sequence
 - `\o{}` Octal escape sequence.

Regex

- In Perl, you can use `\x`
 - | `\x` Unicode "extended grapheme cluster". Not in [].
- You can use Regex ranges with code points
- You might be able to match by Regex classes (Perl, Rust)

```
let re = Regex::new(r"\p{Greek}+").unwrap();
```



<http://www.unicode.org/reports/tr18/>

Emoji

- Combinations or new emoji might not be supported
 - ☐ U+1F92E FACE VOMITTING (Emoji 5.0, 2017)
 - ☐♂ <U+1F937 SHRUG, U+2642 MALE> (Emoji 4.0, 2016)
 -  Ninja Cat riding T-Rex (Windows 10 only)



Emoji

- Replace emoji with images (GitHub, Twitter)
 - <https://github.com/twitter/twemoji>
- Use (coloured) emoji fonts
 - <https://github.com/eosrei/emojione-color-font>
 - <https://github.com/googlei18n/noto-emoji>
- Let it be

Apple



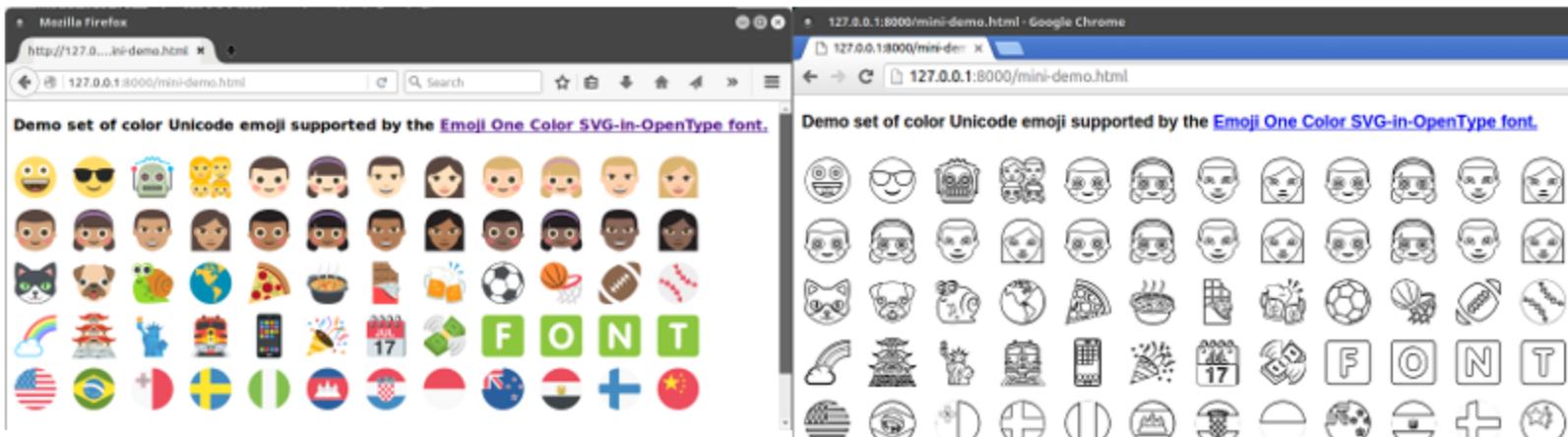
Google



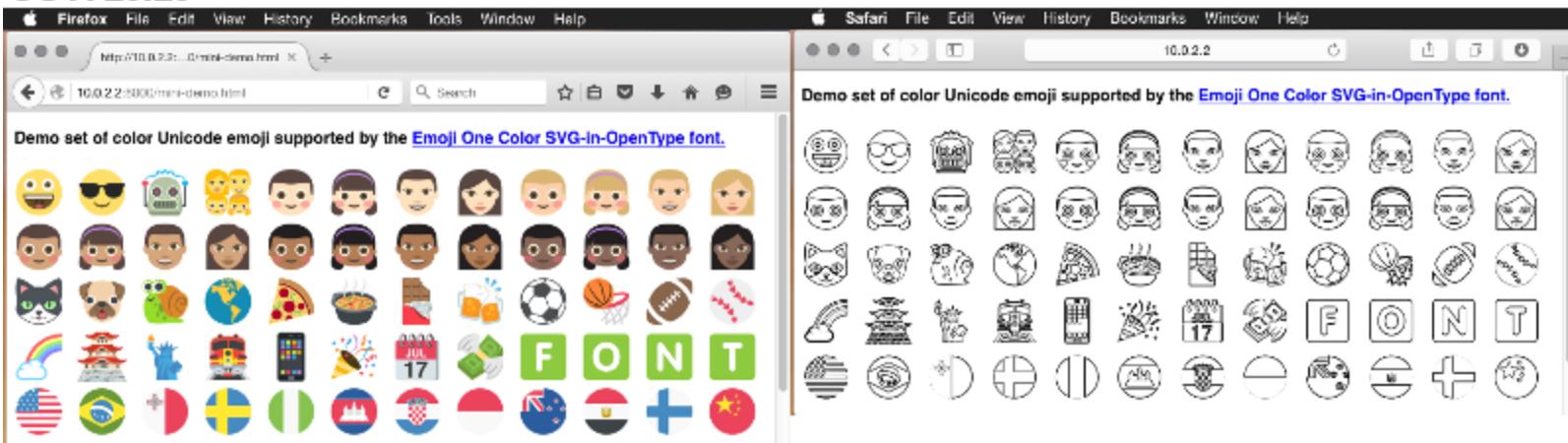
Microsoft



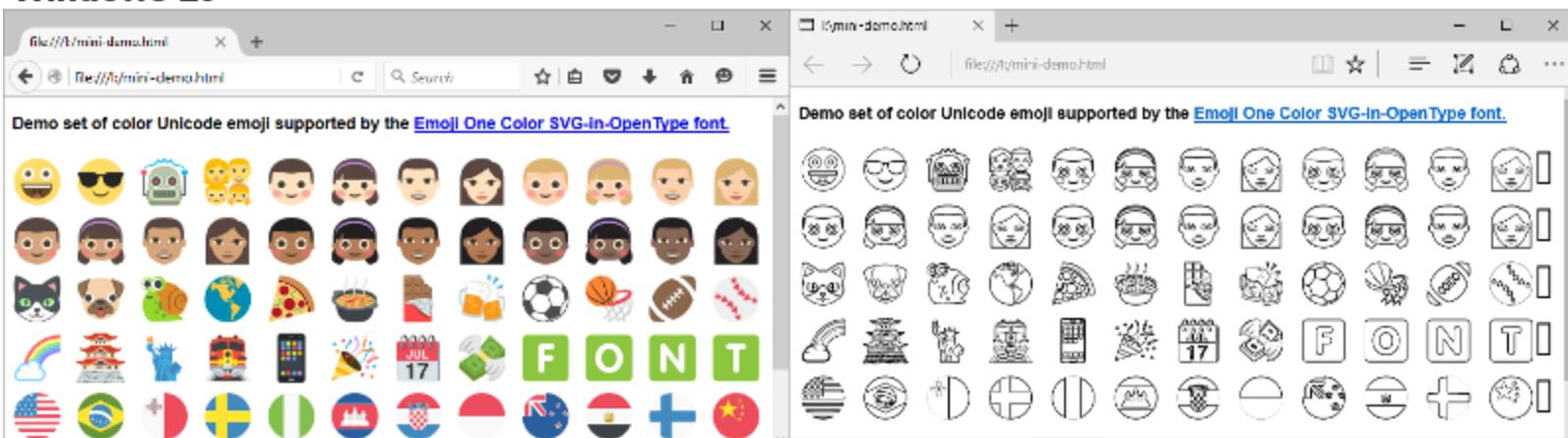
Ubuntu Linux



OS X 10.10



Windows 10



Developing for Unicode

If you ever need to develop Unicode parsing and processing, use the CLDR database:

<http://cldr.unicode.org/>

- * Locale-specific patterns for formatting and parsing: dates, times, numbers, etc.
- * Translations of names: languages, scripts, countries and regions.
- * Language & script information: characters used; plural cases; case mappings.
- * Country information: language usage, currency information, calendar information.
- * Other: ISO & BCP 47 code support (cross mappings, etc.), keyboards, and more.

Security

Read *Unicode Security Considerations*
@ <http://www.unicode.org/reports/tr36/>

Restrict passwords and user names to ASCII

- For logistical reasons (customer support)
- Unicode normalisation of passwords can cause problems
- Equivalent characters

- Basic authentication can fail in different browsers

Sanitise text input

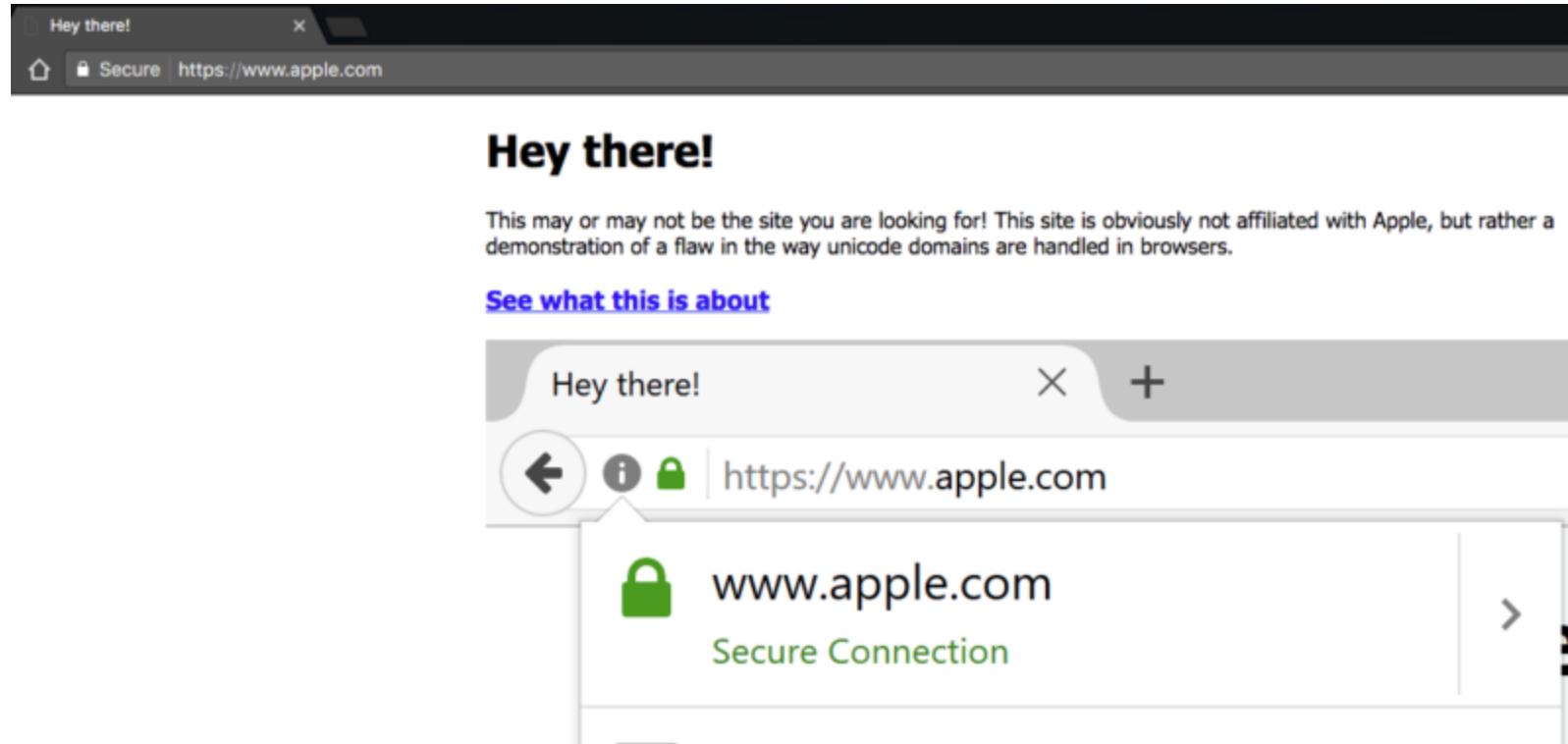
- Difficult problem
- “Unicode injection”: RTL, combining characters, wide characters
- بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ is one (1!) character
U+FDFD ARABIC LIGATURE BISMILLAH AR-RAHMAN AR-RAHEEM
- ZA'LGO!
- 25 different whitespace characters

<https://github.com/minimaxir/big-list-of-naughty-strings>

Unicode in URLs

Visit <https://www.xn--80ak6aa92e.com/> in your browser

Unicode in URLs



Unicode in URLs

<https://www.appIe.com/>

a	U+0430 CYRILLIC SMALL LETTER A
p	U+0440 CYRILLIC SMALL LETTER ER
I	U+04CF CYRILLIC SMALL LETTER PALOCHKA
e	U+0435 CYRILLIC SMALL LETTER IE

<https://www.xudongz.com/blog/2017/idn-phishing/>

Unicode in URLs

- Handing legit Unicode in URLs

`http://Bücher.de`

→ `http://xn--bcher-kva.de`

→ `http://bücher.de`

- Punycode, ASCII representation for Unicode domain names (IDN)

<http://www.unicode.org/reports/tr46/>

Free pizza!

Title: Free Pizza Fridays!

From: HR

To: You

Happy Friday!

Visit <https://tech.gov.sg/free.pizza> to claim a FREE !

FYNAP

- HR

This message could be a scam. [Report] [Ignore]

Unicode in URLs

/ U+2044 FRACTION SLASH

Visit <https://tech.gov.sg/free.pizza> to claim a FREE !



[sg/free.pizza](https://tech.gov.sg/free.pizza) 

Unicode in URLs

Solution: Use Punycode where/when it makes sense to

Visit <https://tech.gov.xn--sgfree-qq0c.pizza> to claim a
FREE !

[Click here](#) for one neat trick to ruin bad software!

- MySQL UTF-8

What happens when the *valid* UTF-8 string



U+1F47D EXTRATERRESTRIAL ALIEN

is inserted into a column of

VARCHAR CHARACTER SET utf8

III-formed sequences and encoding mismatches

- MySQL < 5.5.3 (2010) UTF-8

Incorrect string value: '\xF0\x9F\x91\xBD...' for column 'data' at row 1

In MySQL, use `utfmb4` (\geq 5.5.3, 2010)

<https://mathiasbynens.be/notes/mysql-utf8mb4>

III-formed sequences and encoding mismatches

- ² Python 2

```
>>> '\x81'.decode('utf-8')
# UnicodeDecodeError: 'utf8' codec can't decode byte
# 0x81 in position 0: unexpected code byte
```

-  Ruby 1.9

```
'ü'.encode('ISO-8859-1') + 'ü'
# incompatible character encodings: ISO-8859-1 and
# UTF-8 (Encoding::CompatibilityError)

# or sometimes: invalid multibyte char (US-ASCII)
```

Solution: use languages/libraries which handle Unicode right

Buffer overflows

- Do not assume Unicode strings are of fixed-length

```
Fluß → FLUSS → fluss
```

```
>> '﷽'.length  
1  
  
>> '﷽'.normalize('NFKC').length  
18
```

Solution: use languages/libraries which handle Unicode right

OS/locale filenames

- Beware simple filename sanitisation, especially on Windows
- Normalization of paths
 - c : \w i n d o w s becomes c:\windows
- Character mappings
 - ¥ is mapped to \ on a Japanese-language Windows system

[https://msdn.microsoft.com/en-us/library/dd374047\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/dd374047(v=vs.85).aspx)

```
> 1 + 1;  
← 2  
  
> 1 + 1;  
←  SyntaxError: illegal character 
```

; U+037E GREEK QUESTION MARK

A list of similar characters

Resources

- The Unicode Standard, v10.0 – Core Specification
- Unicode publications
- Unicode technical reports
- Unicode data files
- Unicode public files
- Emoji charts
- Emoji slides
- Unicode character inspector
- UTF-8 decoder
- Big List of Naughty Strings
- Personal names around the world
- Falsehoods Programmers Believe About Phone Numbers
- *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard* by Richard Gillam