Unicode and its ⬬🍁s: programmer essentials and more

⟶

Updated 2020

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

Unicode and its □ 糞 □ 香 s: programmer essentials and m ʌΓore

Shift-JIS edition

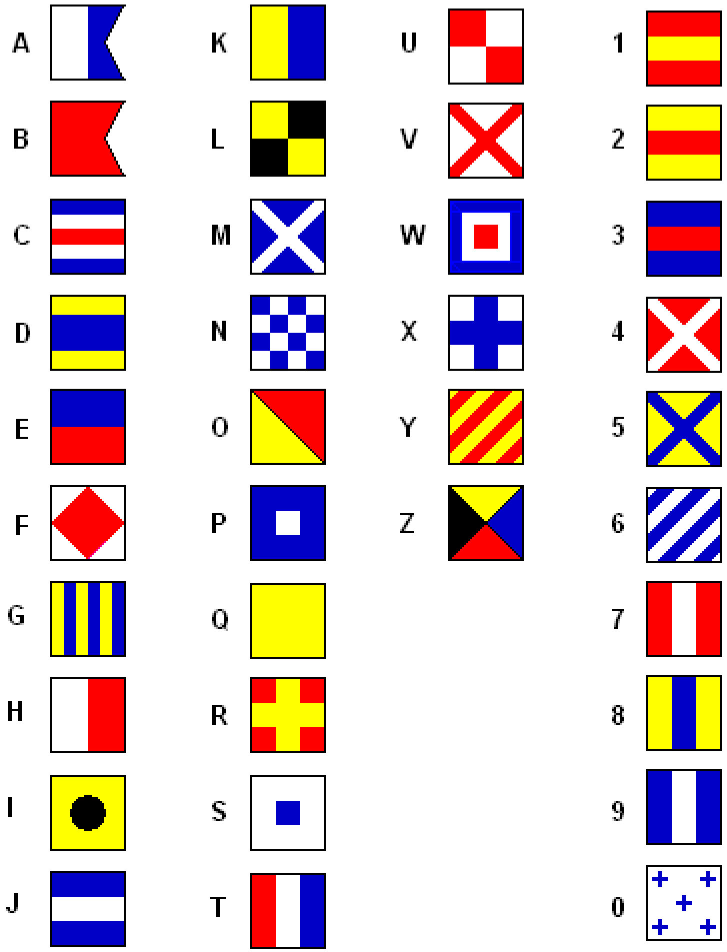
https://github.com/gyng/book/tree/master/slides/unicode

1. History
2. Unicode and UTF-x
3. Programmer pitfalls

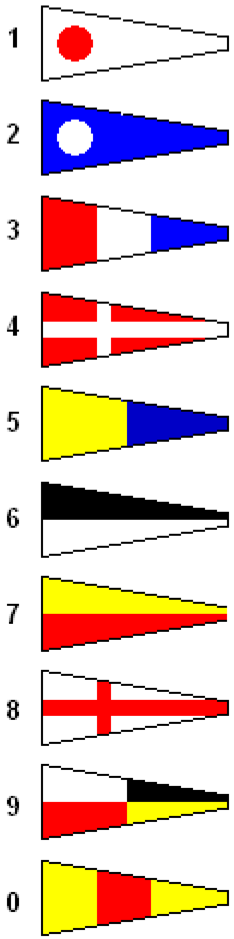
```
> 1 + 1;
← 2

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

Part 1: Encodings

not encryption



Maritime Signal Flags



Penants

```
M   O   R   S  E          C    O   D  E
-- --- ·-· ··· · (space) -·-· --- -·· ·
```

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

丨 丶 丿 乙 亅 二 亠

丈	丰	乏	事	交	仆	伃	伙	住	佶
〇〇〇五 三	〇〇二五 串	〇〇四五 乖	〇〇六五	〇〇八五 亥	〇一〇五 仇	〇一二五 伶	〇一四五 伯	〇一六五 佐	〇一八五 佾
〇〇〇六 上	〇〇二六	〇〇四六 乘	〇〇六六	〇〇八六 亦	〇一〇六 今	〇一二六 仲	〇一四六 估	〇一六六 佑	〇一八六 使
〇〇〇七 下	〇〇二七	〇〇四七	〇〇六七 二	〇〇八七 享	〇一〇七 介	〇一二七 佽	〇一四七 佚	〇一六七 佔	〇一八七 侃
〇〇〇八 不	〇〇二八 丶	〇〇四八	〇〇六八 于	〇〇八八 㐬	〇一〇八 仍	〇一二八 仳	〇一四八 你	〇一六八 何	〇一八八 來
〇〇〇九 丏	〇〇二九 丸	〇〇四九	〇〇六九 云	〇〇八九 亨	〇一〇九 仔	〇一二九 仵	〇一四九 伲	〇一六九 佗	〇一八九 侈
〇〇一〇 丑	〇〇三〇 凡	〇〇五〇 乙	〇〇七〇 互	〇〇九〇 京	〇一一〇 仕	〇一三〇 件	〇一五〇 伴	〇一七〇 佘	〇一九〇 例
〇〇一一 且	〇〇三一 丹	〇〇五一 九	〇〇七一 五	〇〇九一 亭	〇一一一 他	〇一三一 攸	〇一五一 佮	〇一七一 余	〇一九一 侍
〇〇一二 丕	〇〇三二 主	〇〇五二 乞	〇〇七二 井	〇〇九二 亮	〇一一二 仗	〇一三二 价	〇一五二 伸	〇一七二 佛	〇一九二 侏
〇〇一三 世	〇〇三三	〇〇五三 也	〇〇七三 亘	〇〇九三 亳	〇一一三 付	〇一三三 任	〇一五三 伺	〇一七三 作	〇一九三 侐
〇〇一四 丘	〇〇三四	〇〇五四 乩	〇〇七四 亙	〇〇九四 亶	〇一一四 仙	〇一三四 仿	〇一五四 伻	〇一七四 佞	〇一九四 侑
〇〇一五 丙	〇〇三五 丿	〇〇五五 乳	〇〇七五 况	〇〇九五 亹	〇一一五 仝	〇一三五 企	〇一五五 似	〇一七五 佟	〇一九五 侔
〇〇一六 丞	〇〇三六 乂	〇〇五六 乾	〇〇七六 些	〇〇九六	〇一一六 仞	〇一三六 伉	〇一五六 伽	〇一七六 佩	〇一九六 侖
〇〇一七 丟	〇〇三七 乃	〇〇五七 亂	〇〇七七 亞	〇〇九七	〇一一七 仟	〇一三七 伊	〇一五七 佃	〇一七七 佪	〇一九七 侗

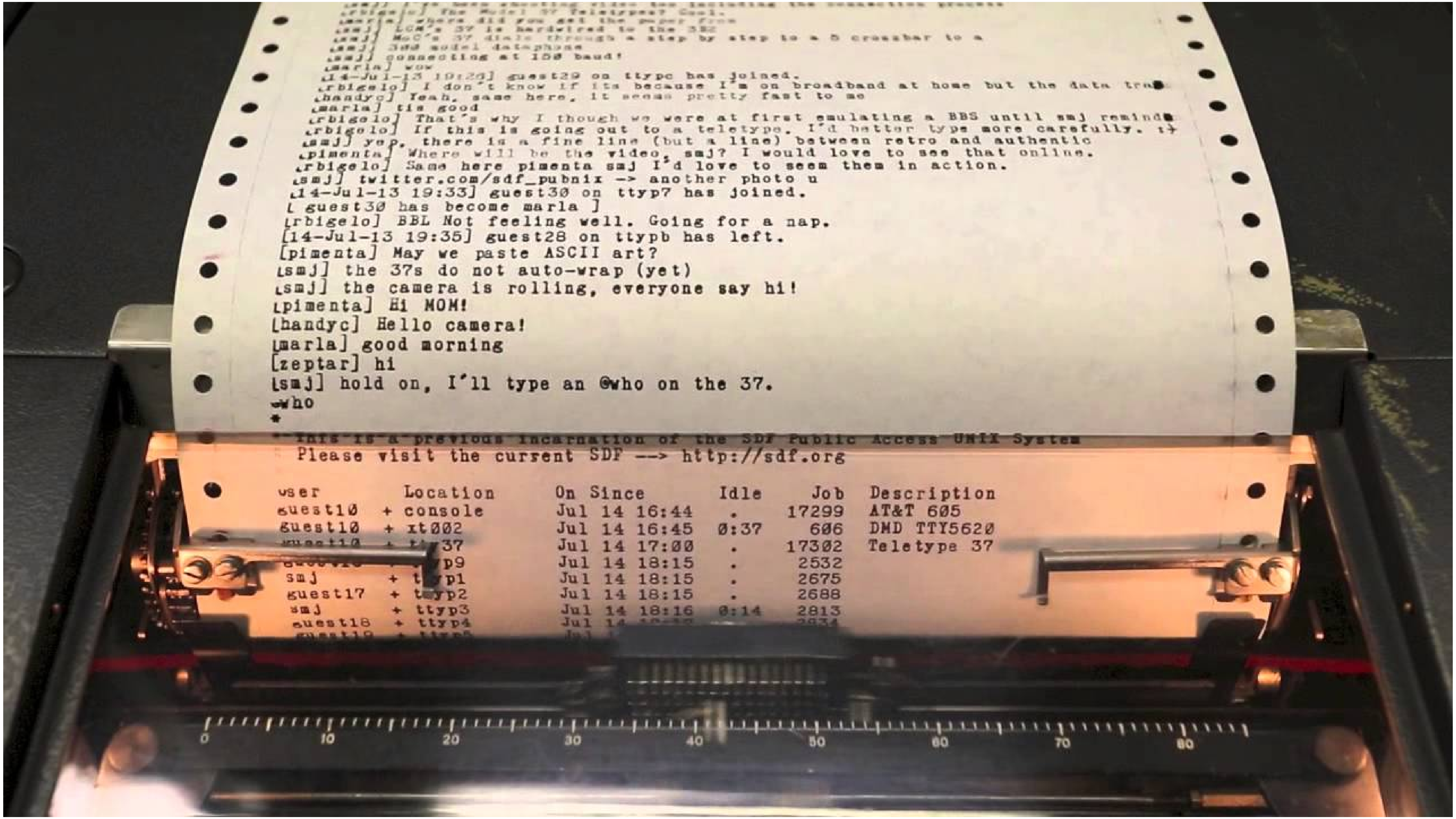
```
電   碼

7193 4316
--... .---- ----. ...-- / ....- ...-- .---- -....

EGL  EWS
. --. .-.. / . .-- ...
```

- Character: `a`, `e`, `1`, `電`, etc.
- Codepoint: mapping of a character to some value
- Encoding: A collection of codepoints

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/



```
[smj] I've been shooting video too including the connection process
[rbigelo] The Model 37 Teletypes? Cool.
[marla] where did you get the paper from
[smj] LCM's 37 is hardwired to the 3B2
[smj] MoC's 37 dials through a step by step to a 5 crossbar to a
[smj] 300 model dataphone
[smj] connecting at 150 baud!
[marla] wow
[14-Jul-13 19:26] guest29 on ttypc has joined.
[rbigelo] I don't know if its because I'm on broadband at home but the data tra
[handyc] Yeah, same here, it seems pretty fast to me
[marla] tis good
[rbigelo] That's why I though we were at first emulating a BBS until smj remind
[rbigelo] If this is going out to a teletype. I'd better type more carefully. :)
[smj] yep, there is a fine line (but a line) between retro and authentic
[pimenta] Where will be the video, smj? I would love to see that online.
[rbigelo] Same here pimenta smj I'd love to seem them in action.
[smj] twitter.com/sdf_pubnix -> another photo u
[14-Jul-13 19:33] guest30 on ttyp7 has joined.
[ guest30 has become marla ]
[rbigelo] BBL Not feeling well. Going for a nap.
[14-Jul-13 19:35] guest28 on ttypb has left.
[pimenta] May we paste ASCII art?
[smj] the 37s do not auto-wrap (yet)
[smj] the camera is rolling, everyone say hi!
[pimenta] Hi MOM!
[handyc] Hello camera!
[marla] good morning
[zeptar] hi
[smj] hold on, I'll type an @who on the 37.
@who
*
**This is a previous incarnation of the SDF Public Access UNIX System
  Please visit the current SDF --> http://sdf.org

user        Location       On Since       Idle    Job  Description
guest10   + console        Jul 14 16:44     .   17299  AT&T 605
guest10   + xt002          Jul 14 16:45   0:37    606  DMD TTY5620
guest10   + tty37          Jul 14 17:00     .   17302  Teletype 37
guest10   + ttyp9          Jul 14 18:15     .    2532
smj       + ttyp1          Jul 14 18:15     .    2675
guest17   + ttyp2          Jul 14 18:15     .    2688
smj       + ttyp3          Jul 14 18:16   0:14   2813
guest18   + ttyp4          Jul 14 18:[illegible]     2934
guest19   + ttyp5          Jul 1[illegible]
```

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

ASCII

- 0–31 are control characters `NUL` `CR` `LF` `DEL`
- 32–126 are punctuation, numerals and letters
- `␣` in binary: `0100000` $= 32 = \text{0x20}$
- `A` in binary: `1000001` $= 65 = \text{0x41}$
- `a` in binary: `1100001` $= 97 = \text{0x61}$
    - $= 65 + 32$

      _ $= \text{0x41} + \text{0x20}$

      _ $=$ `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)

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

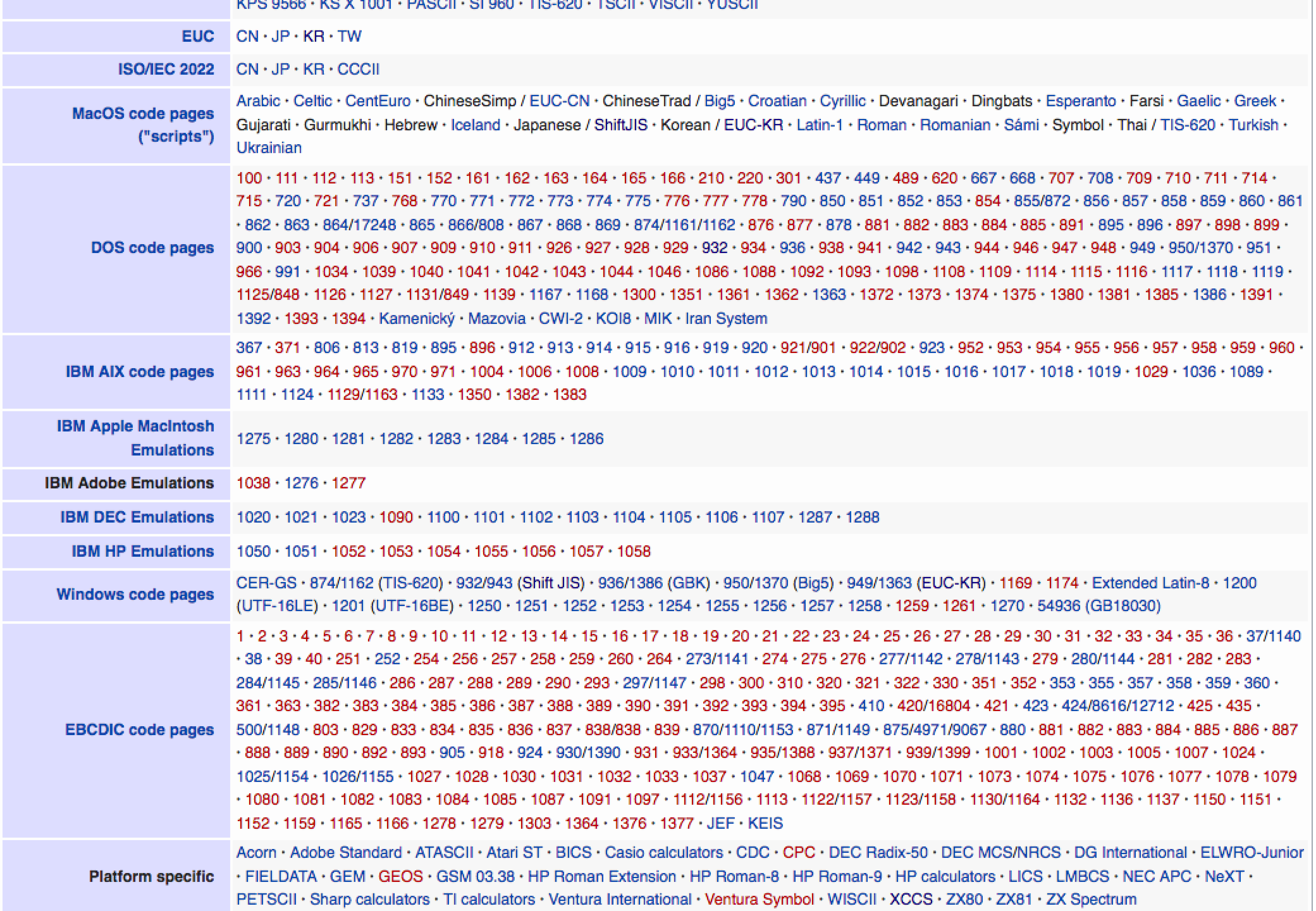
- Latin-centric
- Everybody else came up with their own encodings
- Alternative ASCII sets cause problems with interchange

moji bake

- Mojibake (文字化け): JIS, Shift-JIS, EUC, and Unicode
- No emoji, only emoticons :-(

Dark ages

- ???
- ???
- ???
- ???
- ???
- ???



	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

Part 2: 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)

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 13.0 (2020 March 10)

> Unicode 13.0 adds 5,930 characters, for a total of 143,859 characters.

https://unicode.org/versions/Unicode13.0.0/

> 55 new emoji characters

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

...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 = Unicode Transformation Format)

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

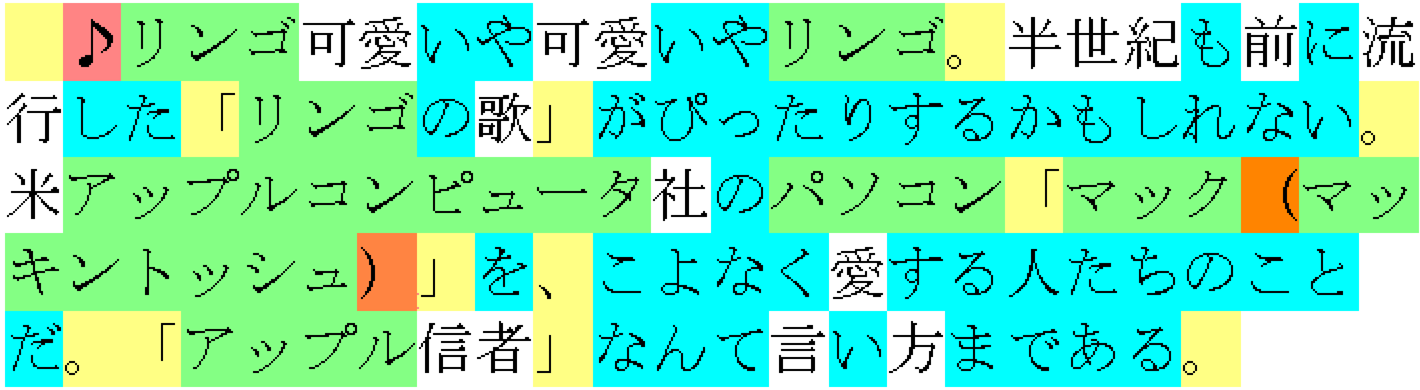
SCSU

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（マッキントッシュ）」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

(The good one)

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`

Unicode features

Combining characters

- Modify other characters

  `e` + `´` = `é`

  `<e U+0065 LATIN SMALL LETTER E,`

  `U+0301 COMBINING ACUTE ACCENT>`

- Modifiers come after base character

Combining characters

- Precomposed `é`

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

`é` ≠ `é`?

`é` = `é`?

Unicode normalisation

- Some combined characters are the same, sometimes

Unicode normalisation mumbo jumbo

- Equivalence criteria
  - canonical (NF)
  - compatibility (NFK)
- `ﬃ` `U+FB03 LATIN SMALL LIGATURE FFI` vs `f` `f` `i`
  - not equivalent under canonical (NF)
  - equivalent under NFK compatiability (NFK)

Unicode normalisation

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

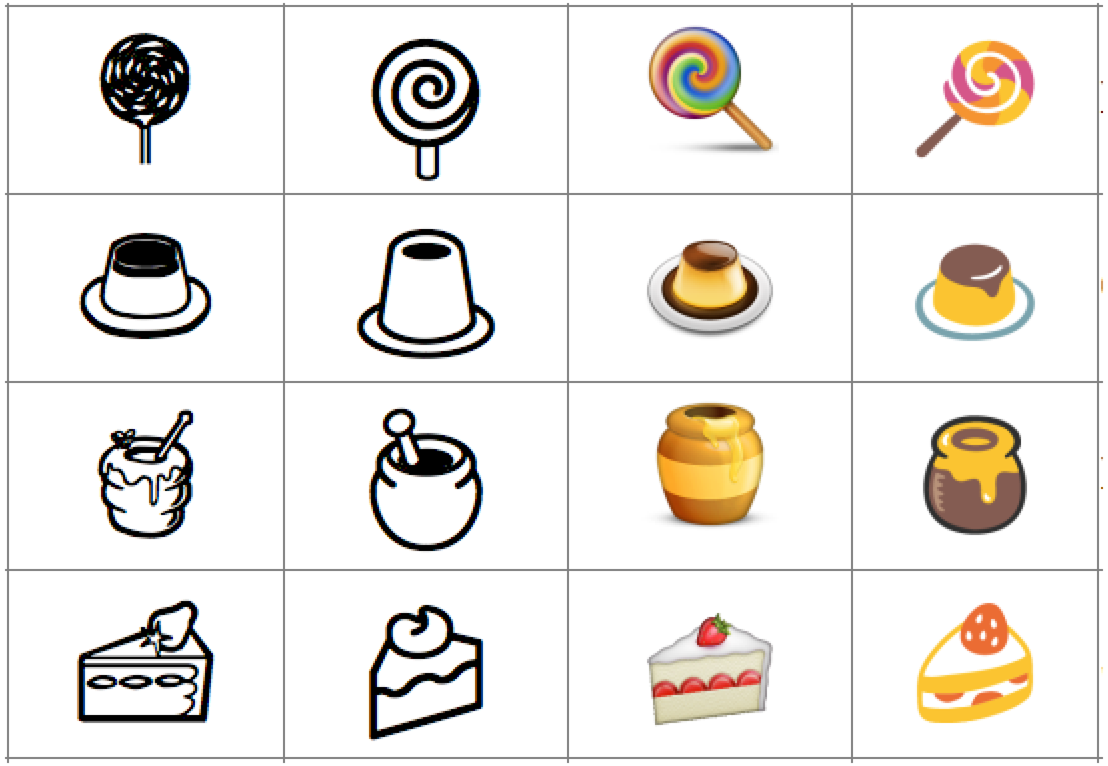
NF is used to canonicalise combining characters

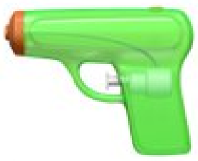
Emoji

- $\overset{\text{e}}{\text{絵}}$ (≅ picture) ＋ $\overset{\text{moji}}{\text{文字}}$(≅ written character)
- Early emoji were created by Japanese telcos
- 2008: Gmail, iPhone
- 2010: Unicode 6
- 🈲 🈳 🈴 🈵 🈶 🈷 🈸 🈹 🈺 🆖 🆗 🉑 🈁 🈂 🌊 🎌 🎏

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

Can be represented differently





iOS

Android

Windows

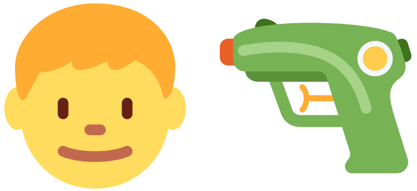
Samsung

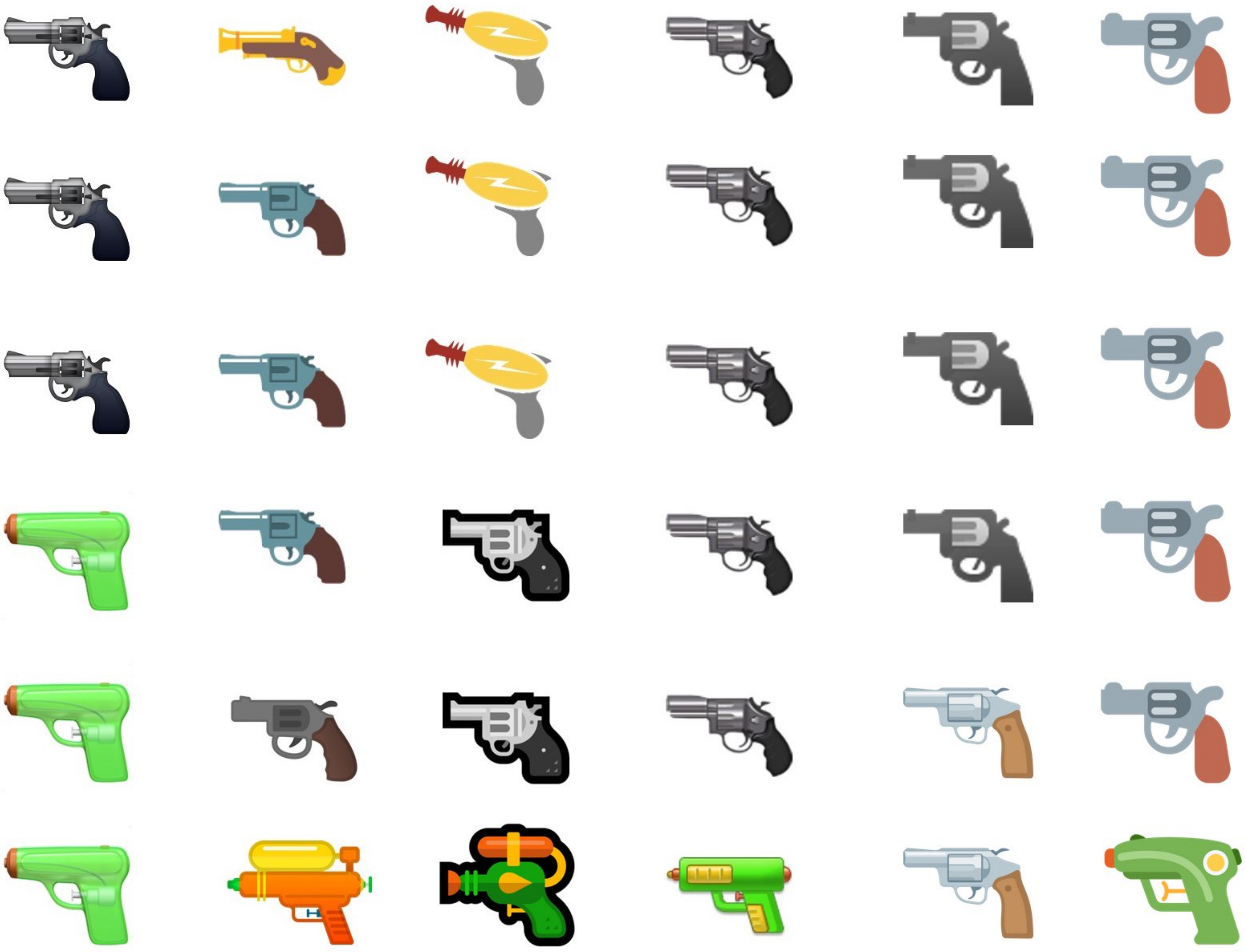
LG

HTC

Facebook

Twitter





Apple Google Microsoft Samsung Facebook Twitter

2013

2014

2015

2016

2017

2018

Combining emoji

👨 + 👩 + 👦 = 👨‍👩‍👦

👪 ≠ 👨‍👩‍👦

👪 `U+1F46A FAMILY` vs combined character

🇸 + 🇬 = 🇸🇬

🇬 + 🇸 = 🇬🇸

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

Variation selectors

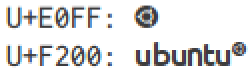
🏻 🏼 🏽 🏾 🏿

👦 👦🏻 👦🏼 👦🏽 👦🏾 👦🏿

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

Private use areas

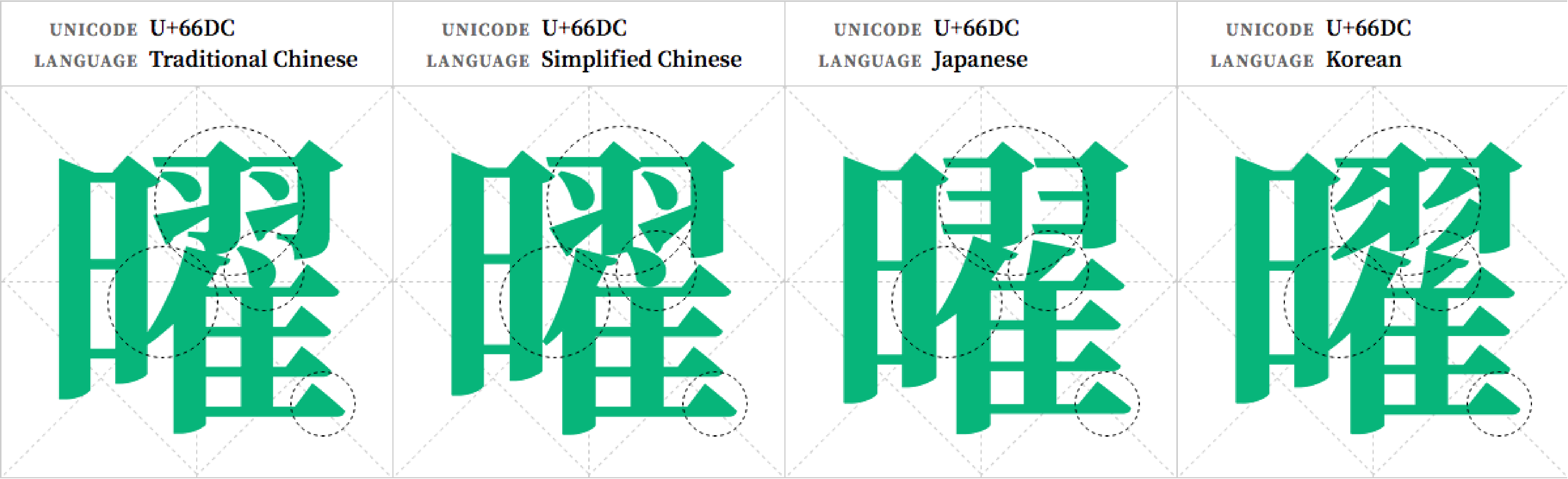
- `U+E000..U+F8FF`, `U+F0000..U+FFFFD`, `U+100000..U+10FFFD`
- Suggested for internal use
  - data processing
  - artificial scripts
  - ancient scripts
- ð `U+F8FF` (`⇧`-`⌥`-`k`)
- Ubuntu has `U+E0FF` and `U+F200`



```
U+E0FF:
U+F200: ubuntu®
```

Han unification

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

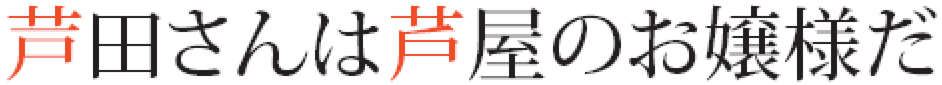


- Different countries have different standards

Han unification

- Variants can be significant (names)

  あ し
  `芦` Ashi·da, given name vs Ashi·ya, old place name

  芦田さんは芦屋のお嬢様だ

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

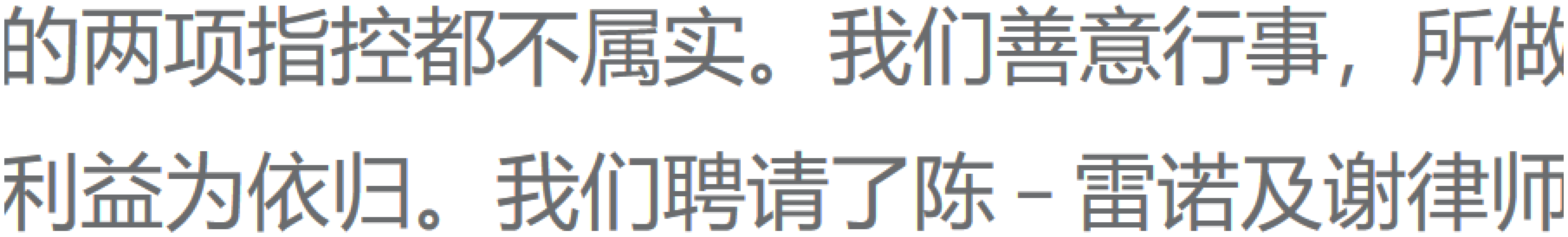
- Lose round-trip conversion compatibility with character sets which have variants

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

Rendering issues

What could possibly go wrong?

`lang="zh"`



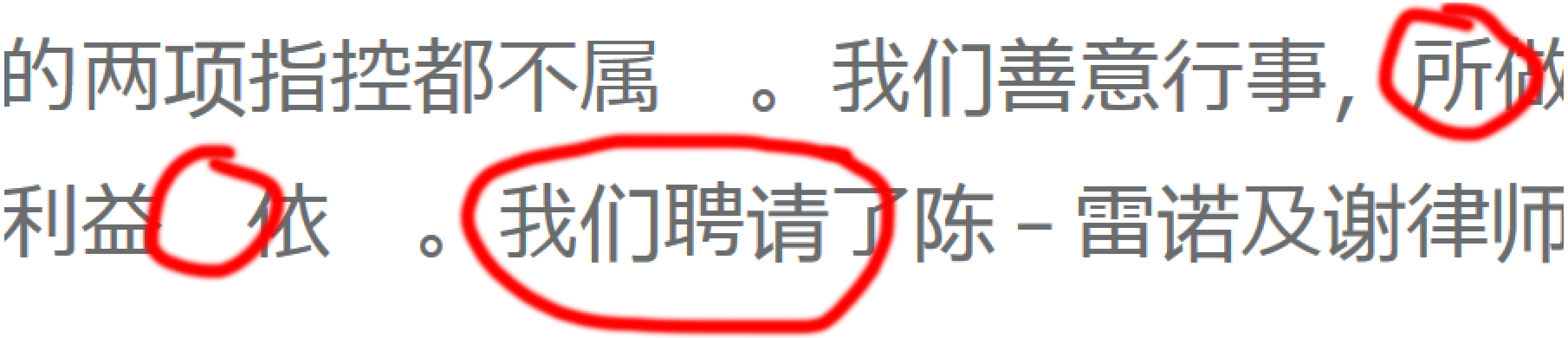
的两项指控都不属实。我们善意行事，所做

利益为依归。我们聘请了陈 – 雷诺及谢律师

Rendering issues

Blank characters, mixed fonts, wrong glyphs

`lang="en"`



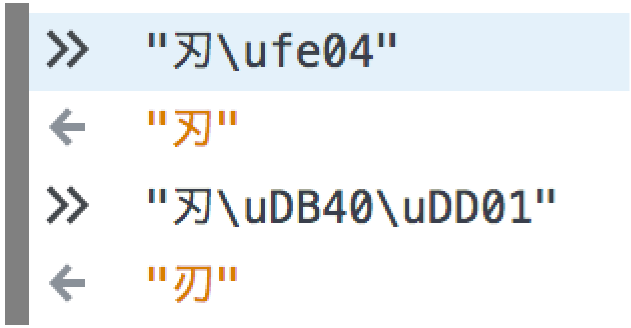
的两项指控都不属　。我们善意行事，所做

利益　依　。我们聘请了陈 – 雷诺及谢律师

Variation selectors

- 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/

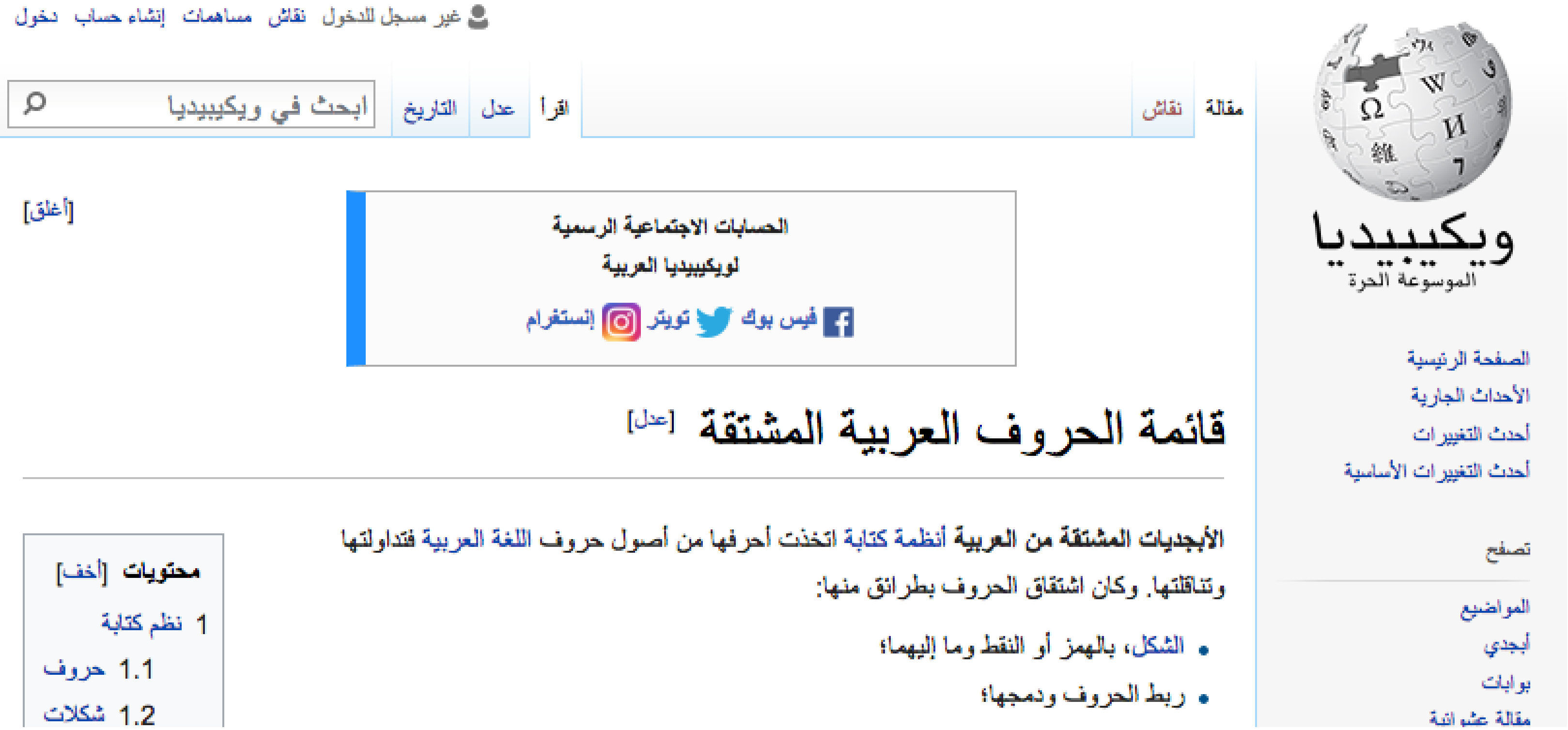
Ligatures

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

- But! There are some predefined ligatures
  `ﬄ U+FB04 LATIN SMALL LIGATURE FFL`
  `Ꜹ U+A738 LATIN CAPITAL LETTER AV`
  `æ U+00E6 LATIN SMALL LETTER AE`
- Similar issue with subscript and superscript

Control sequences and vertical text

- Vertical text
- RTL mark



غير مسجل للدخول نقاش مساهمات إنشاء حساب دخول

مقالة نقاش اقرأ عدل التاريخ ابحث في ويكيبيديا

ويكيبيديا
الموسوعة الحرة

الصفحة الرئيسية
الأحداث الجارية
أحدث التغييرات
أحدث التغييرات الأساسية

تصفح

المواضيع
أبجدي
بوابات
مقالة عشوائية

[أغلق]

الحسابات الاجتماعية الرسمية
لويكيبيديا العربية
فيس بوك تويتر إنستغرام

قائمة الحروف العربية المشتقة [عدل]

محتويات [أخف]
1 نظم كتابة
1.1 حروف
1.2 شكلات

الأبجديات المشتقة من العربية أنظمة كتابة اتخذت أحرفها من أصول حروف اللغة العربية فتداولتها وتناقلتها. وكان اشتقاق الحروف بطرائق منها:

- الشكل، بالهمز أو النقط وما إليهما؛
- ربط الحروف ودمجها؛

Unicode Bidirectional Algorithm @ http://unicode.org/reports/tr9/

Unicode Vertical Text Layout @ http://www.unicode.org/reports/tr50/

EarthWeb commercial, 2001

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

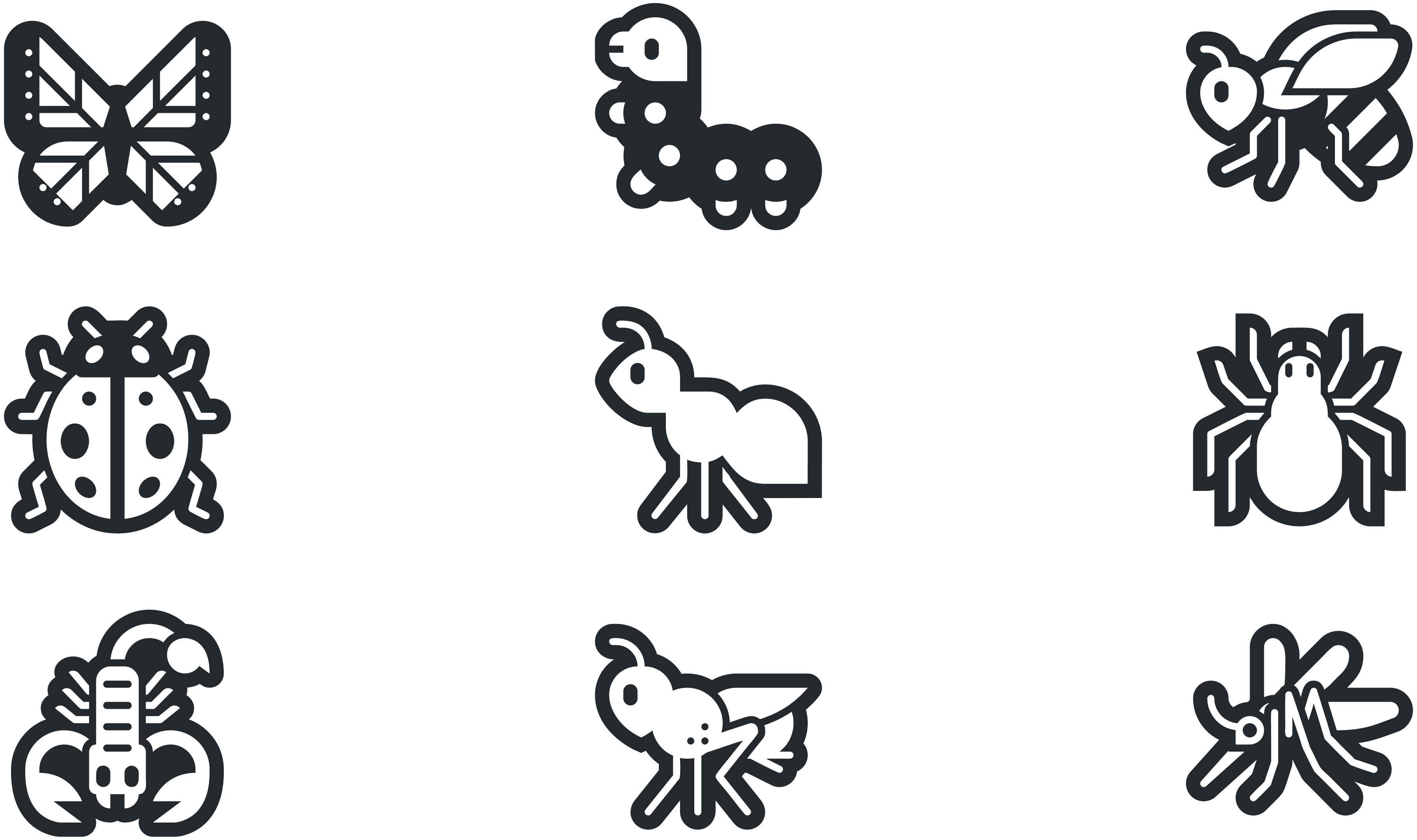
👨‍💻

Part 3: Necessary

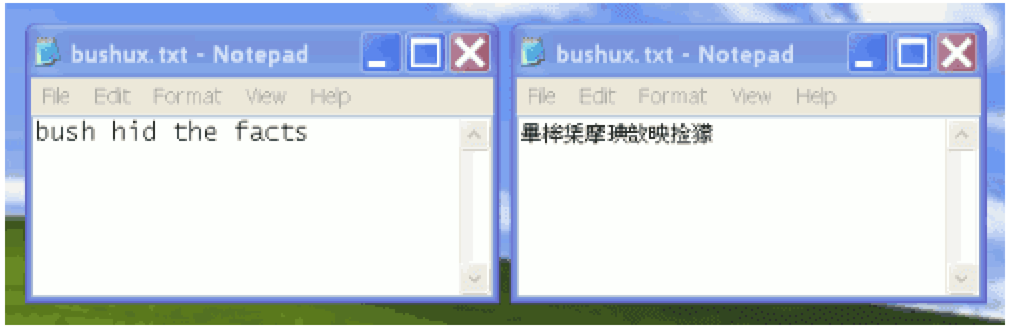
but not necessarily sufficient

programmer knowledge

👩‍💻



“Bush hid the facts”



bush hid the facts

畢桳栠摩琠敨映捡獴

1. Type "Bush hid the facts"
2. Save the file
3. Open the file

https://en.wikipedia.org/wiki/Bush_hid_the_facts

`IsTextUnicode`

> Determines if a buffer is likely to contain a form of Unicode text.

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╨п╫п╥я▐п╞я─я▀
```

UTF-8

```
Библиотека
```

Use UTF-8 for all source code if possible

- Configure your text editor

Magic comments for some older languages

💎 Ruby $\leq$ 1.9.x

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

🐍^{2} Python 2

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

💾 C ≤ C99

```
/* Dear future programmer: Good luck 👍 */
```

(Use a library, utf8proc seems to be popular)

Text processing

- Treat input as bytes (if possible)

Text processing

- Treat output as strings (and not byte arrays)

Text processing

- Use UTF-8 wherever possible

Text processing

- Decide what to do with invalid bytes
  - discard or substitute?
- Do not self-roll your own text encoding library

Log streaming

```
with open("mobydick-emoji-edition-🐳.utf8.txt", "rb") as input:
    while True:
        output_chunk = input.read(4096)
        if not output_chunk:
            #  EOF
            break
        #  Yield each chunk
        yield output_chunk
```

Where's the bug?

Log streaming

```
with open("mobydick-emoji-edition-🐳.utf8.txt", "rb") as input:  # 🐞
    while True:
        output_chunk = input.read(4096)  # 🐞
        if not output_chunk:
            #  EOF
            break
        #  yield each chunk
        yield output_chunk
```

Read in text with the right encoding

Especially when parsing HTML or XML

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

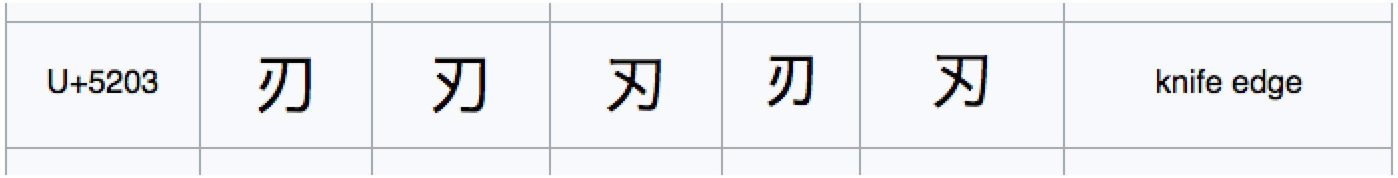
```
# Beautiful Soup
soup = BeautifulSoup(html, fromEncoding='Shift_JIS')
```

Set HTML `charset`

```html
<!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

Use `accept-charset` in forms as needed

```html
<form action="myform" accept-charset="UTF-8">
```

Uses document charset by default

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?

  `ı` → `I`
  `i` → `İ`

Case conversion

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

  `ı` → `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

- 🇩🇪 German
- `ß` upcases to `ss`

Case conversion

- 🇩🇪 German
- `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' // (unchanged)
```

🔥 JavaScript (Firefox 73)

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

🔮 JavaScript (Chrome 59)

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

Python 2

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

Python 3

```
>>> 'ß'.upper()
'SS'
```

💎 Ruby 2.3

```
> "\u{00df}".upcase
=> "ß" # (unchanged)
```

💎 Ruby 2.4

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

🐻 Go

```go
package main

import (
        "fmt"
        "golang.org/x/text/cases"
        "golang.org/x/text/language"
)

func main() {
        c := cases.Upper(language.German)
        fmt.Println(c.String("ß"))
}

SS
```

☕ Java

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

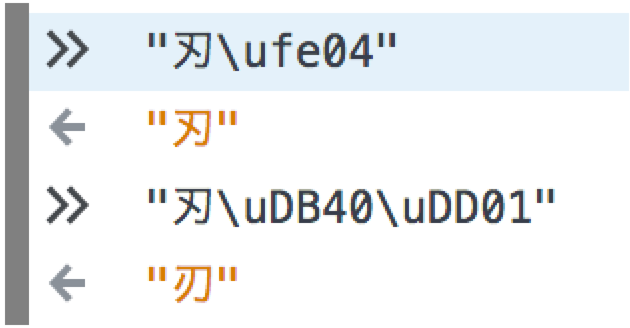
```
SS
```

Rust

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

SS
```

Use variation selectors as needed

`U+E0101 VARIATION-SELECTOR-18`



```
>> "刄\ufe04"
←  "刄"
>> "刄\uDB40\uDD01"
←  "刃"
```

Use a correct font for the language outside HTML

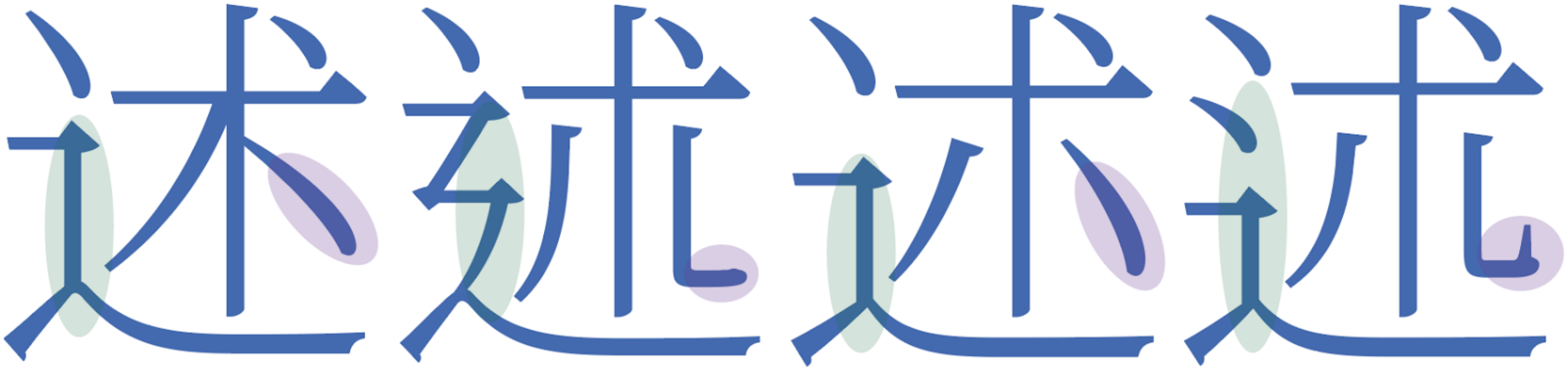
- Google’s Noto/Noto CJK has great support
- Similarly, Adobe’s Source Han

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

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

Use a correct font for the language outside HTML

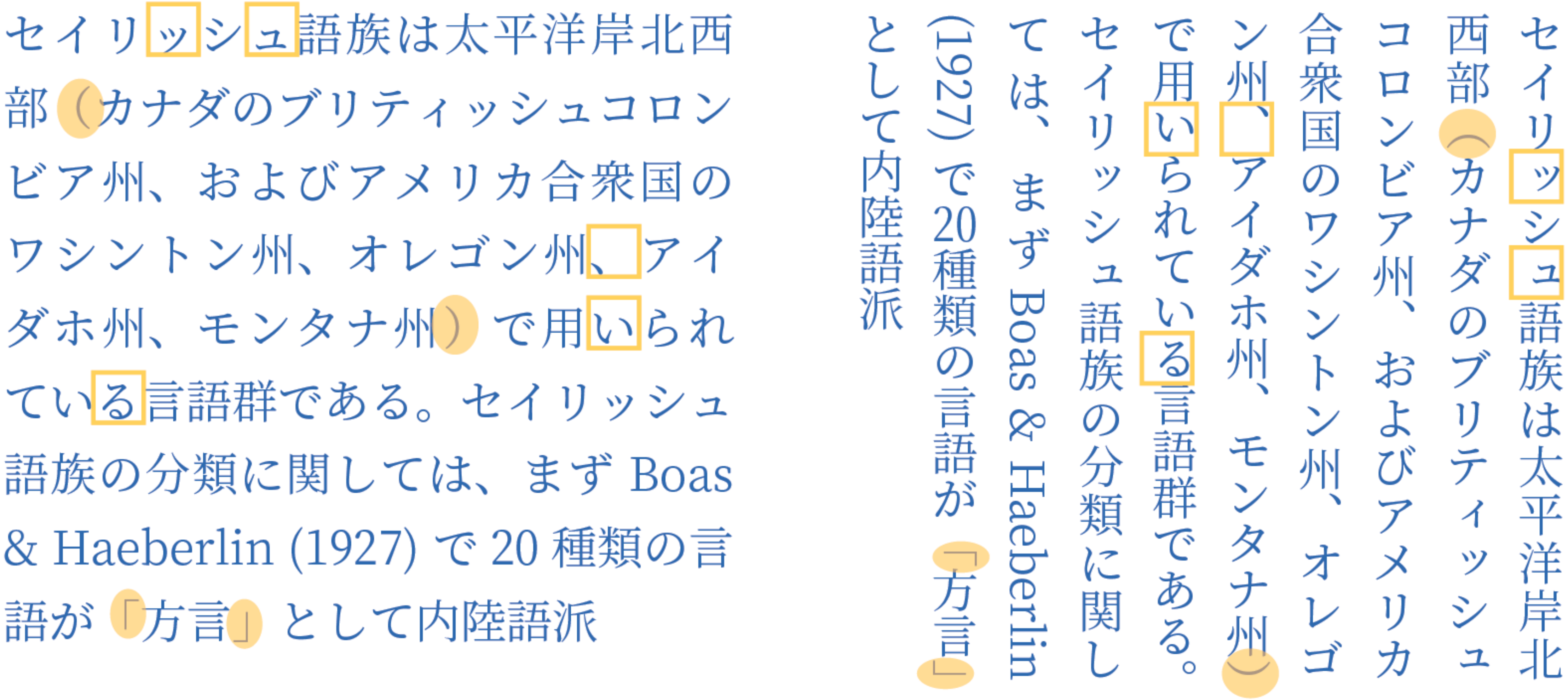
Glyph variations



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

Noto Serif CJK

Vertical text support



セイリッシュ語族は太平洋岸北西部（カナダのブリティッシュコロンビア州、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

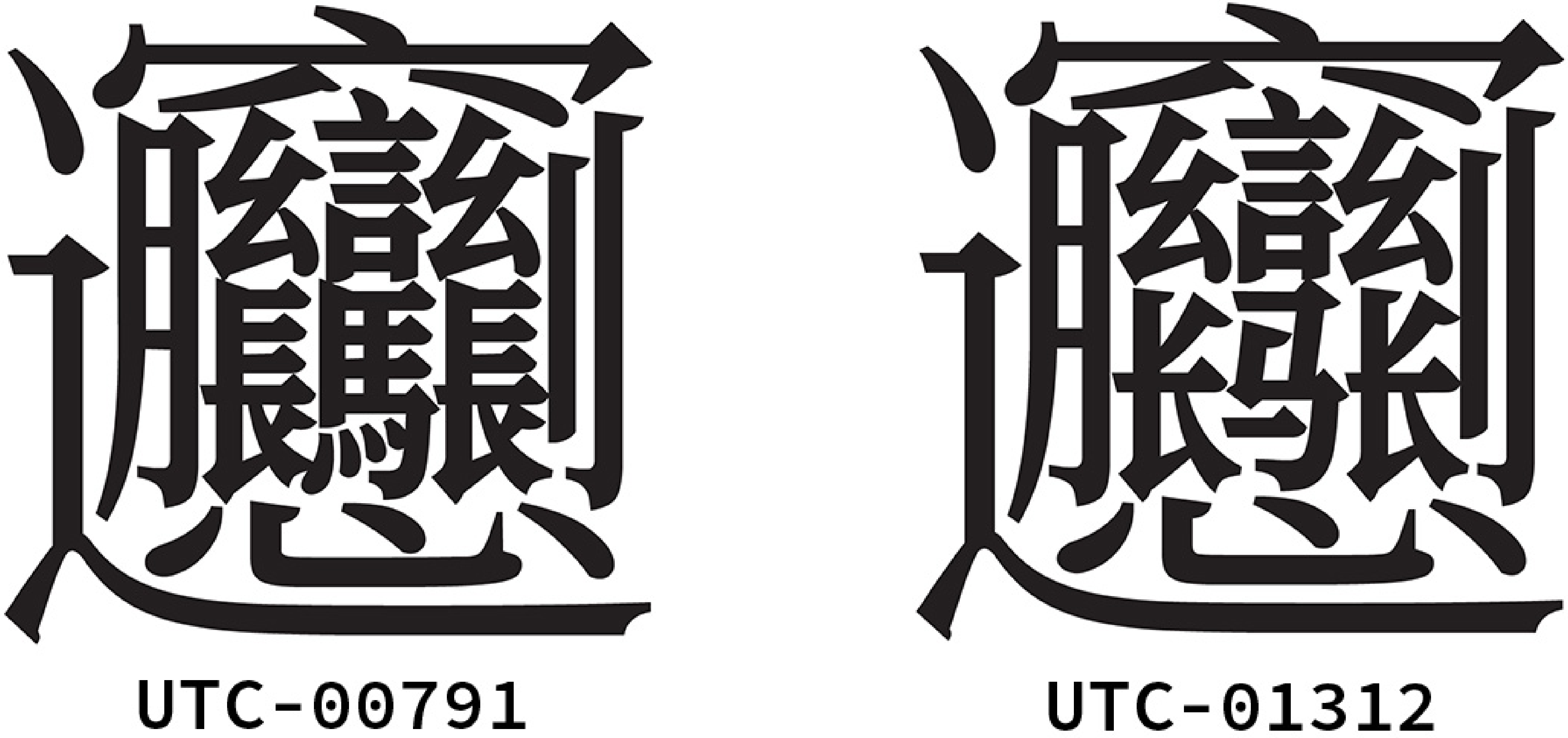
セイリッシュ語族は太平洋岸北西部（カナダのブリティッシュコロンビア州、およびアメリカ合衆国のワシントン州、オレゴン州、アイダホ州、モンタナ州）で用いられている言語群である。セイリッシュ語族の分類に関しては、まずBoas & Haeberlin (1927)で20種類の言語が「方言」として内陸語派

Noto Serif CJK

https://helpx.adobe.com/photoshop/user-guide.html?

Unencoded characters

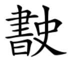
How can I display (CJK/my own) characters not encoded in Unicode?



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

Coming to a Unicode version soon?

Unencoded characters

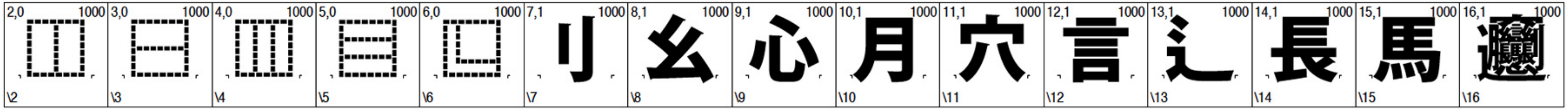
- Use an image
- Use Ideographic Description Sequences
  `⿰⿱⿲⿳⿴⿵⿶⿷⿸⿹⿺⿻ U+2FF0..U+2FFF`
  `⿰書史` for , a character of a dialect in China
- Use fonts which have the unencoded glyph either
  - as an existing character (Wingdings 🕴👽👓✇)
  - in Private Use Area
  - as a combined sequence

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/CCJKType/2014/03/ids-opentype.html

Unencoded characters


2,0 1000 \2
3,0 1000 \3
4,0 1000 \4
5,0 1000 \5
6,0 1000 \6
7,1 1000 刂 \7
8,1 1000 幺 \8
9,1 1000 心 \9
10,1 1000 月 \10
11,1 1000 穴 \11
12,1 1000 言 \12
13,1 1000 辶 \13
14,1 1000 長 \14
15,1 1000 馬 \15
16,1 1000 \16

```
⿺辶⿳穴⿰月⿰⿲⿱幺長⿱言馬⿱幺長刂心 (traditional)
⿺辶⿳穴⿰月⿰⿲⿱幺长⿱言马⿱幺长刂心 (simplified)
```

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

What ⿺辶 ⿳穴 ⿲月 ⿱⿲幺言幺 ⿲長馬長刂心 ⿺辶 ⿳穴 ⿲月 ⿱⿲幺言幺 ⿲長馬長刂心面 looks like

String sorting

```
>> 'e' > 'f'
false

>> 'f' > 'e'
true
```

String comparison done in lexicographical order in JavaScript

String sorting

- Sorting strings is hard!

```
>> 'é' > 'f'
true
```

String sorting

- The solution: normalisation

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

String sorting

- Sometimes

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

MDN: String.prototype.normalize()

https://unicode.org/reports/tr10/#Hangul_Collation

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

Concise bedtime reading

> The essential problem results from the fact that Hangul syllables can also be represented with a sequence of conjoining jamo characters and because syllables represented that way may be of different lengths, with or without a trailing consonant jamo.

Asymmetric searching

query	matches
resume	resume, Resume, RESUME, résumé, rèsumè, 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

What's the length of `café`?

String length

Problems arise when your string contains

- combining marks
- surrogate pairs (UTF-16)

String length — combined characters

```
>> 'café'.length
5

>> 'café'.normalize().length
4
```

```
>> 'ユニコード'.length
5

>> 'ユニコート\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`

🔥 JavaScript

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

🐍^{2} Python 2

```
>>> len(u'💩')
2
```

🐍^{3} Python 3

```
>>> len('💩')
1
```

🐻 Go

```go
fmt.Println(len("💩")) // 4

fmt.Println(len([]rune("💩"))) // 1
```

💎 Ruby

```
>> '💩'.length
1
```

⚙️ Rust

```rust
println!("{}", "💩".len());
// 4

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

☕ Java

```java
System.out.println("💩".length());
// 2

String s = "💩";
System.out.print(s.codePointCount(0, s.length()));
// 1
```

String lengths

- What is the definition of the length of a string?
- What is a character?

String lengths

- Bytes
- Codepoints
- Normalised codepoints
- ~~Characters~~

```
fmt.Println(len([]rune("🏴")))  // => 4
```

Characters

- Do not think about strings in terms of characters
- Characters are not your friends

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();
```

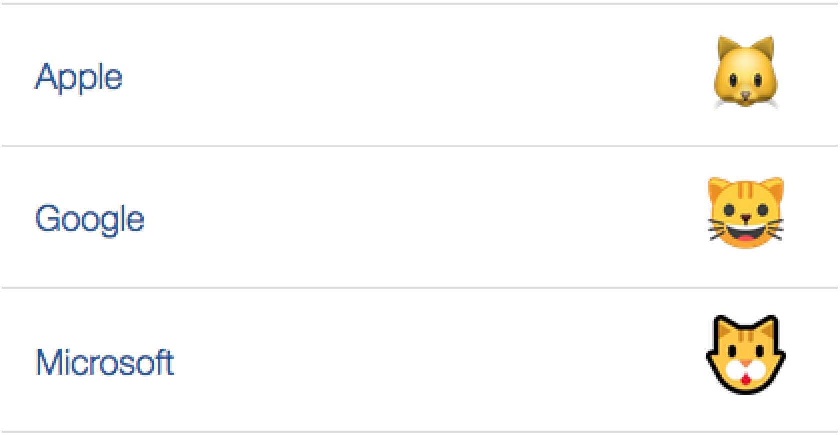
http://www.unicode.org/reports/tr18/

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)

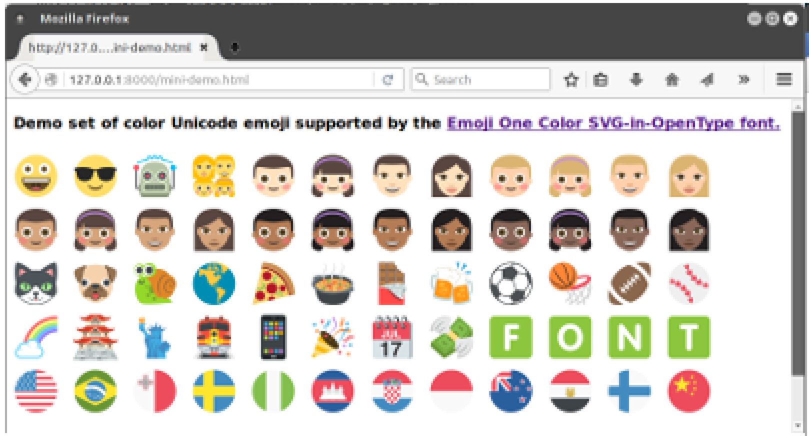
Emoji

- 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

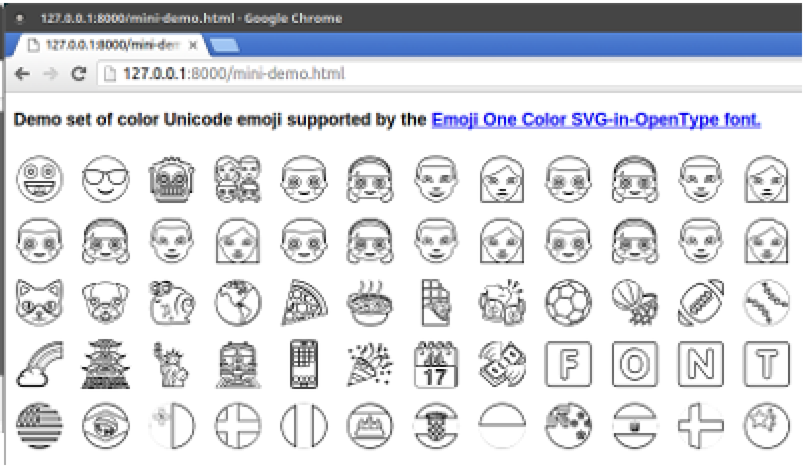


Apple	🐱
Google	🐱
Microsoft	🐱

Ubuntu Linux



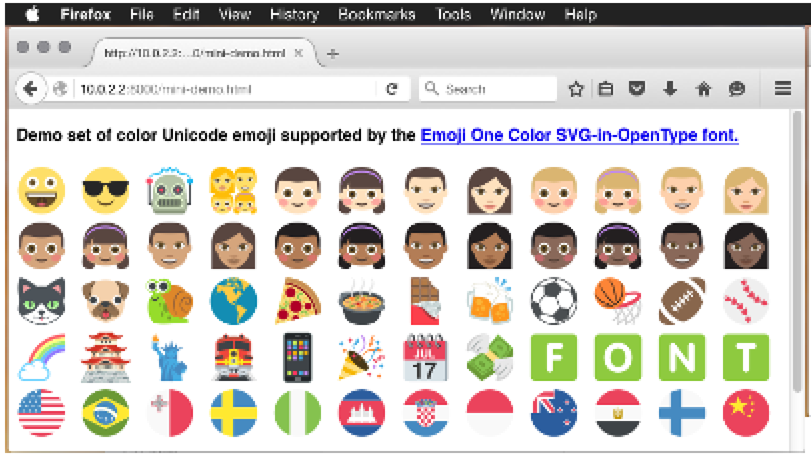
Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.



127.0.0.1:8000/mini-demo.html - Google Chrome
127.0.0.1:8000/mini-de
127.0.0.1:8000/mini-demo.html

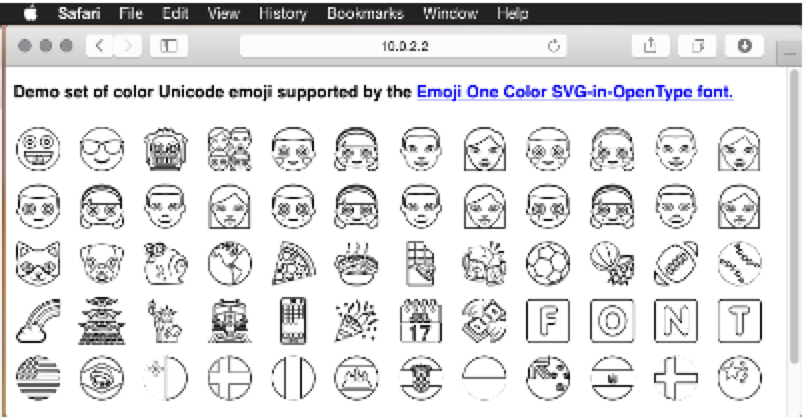
Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.

OS X 10.10



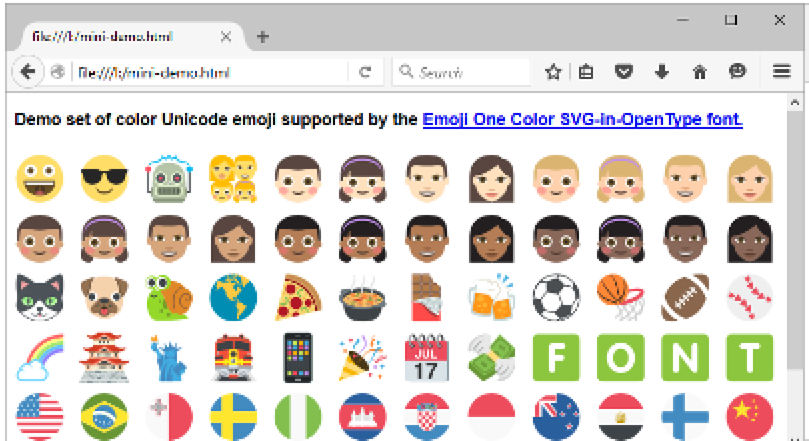
Firefox
File
Edit
View
History
Bookmarks
Tools
Window
Help
http://10.0.2.2:...0/mini-demo.html
10.0.2.2:8000/mini-demo.html
Search

Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.

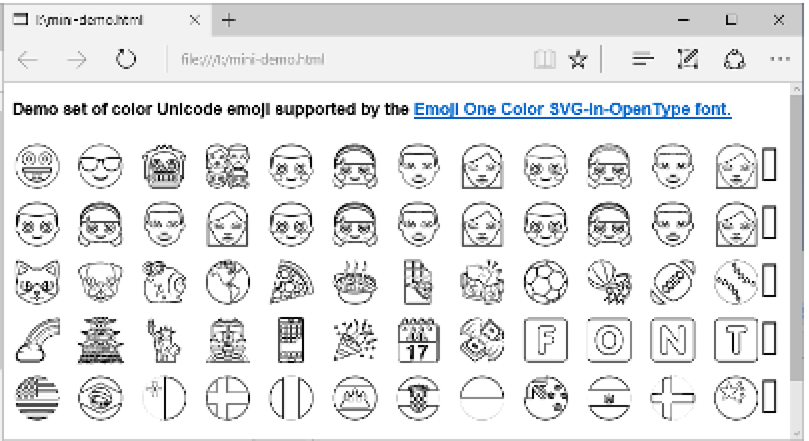


Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.

Windows 10

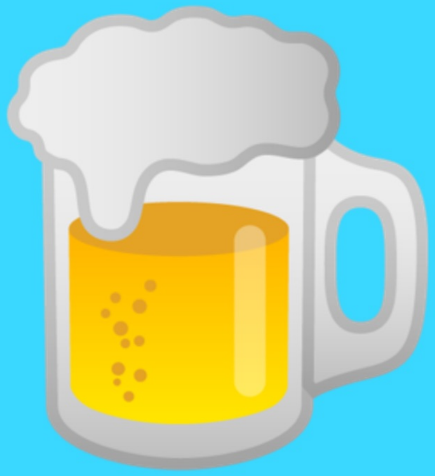


Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.

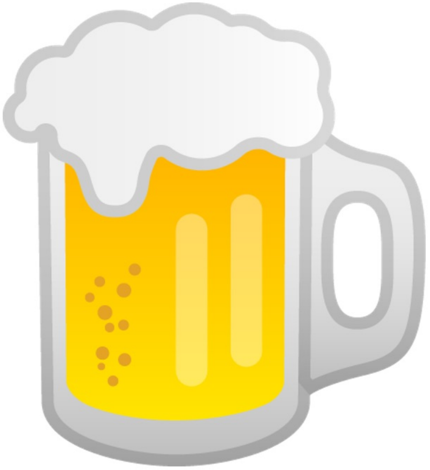


Demo set of color Unicode emoji supported by the Emoji One Color SVG-in-OpenType font.

Emoji bugs



Android 8.0



Android 8.1

https://blog.emojipedia.org/google-fixes-burger-emoji/

Developing for Unicode

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

http://cldr.unicode.org/

Email

Set the MIME needed for Unicode if your library doesn't handle it for you

```
msg = MIMEText('€10'.encode('utf-8'), _charset='utf-8')
```

https://docs.python.org/3.1/library/email.mime.html#email.mime.text.MIMEText

https://en.wikipedia.org/wiki/Unicode_and_email

🚨 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
  `e` $+$ `` $\neq$ `é`
- Basic authentication can fail in different browsers
- Keyboard issues

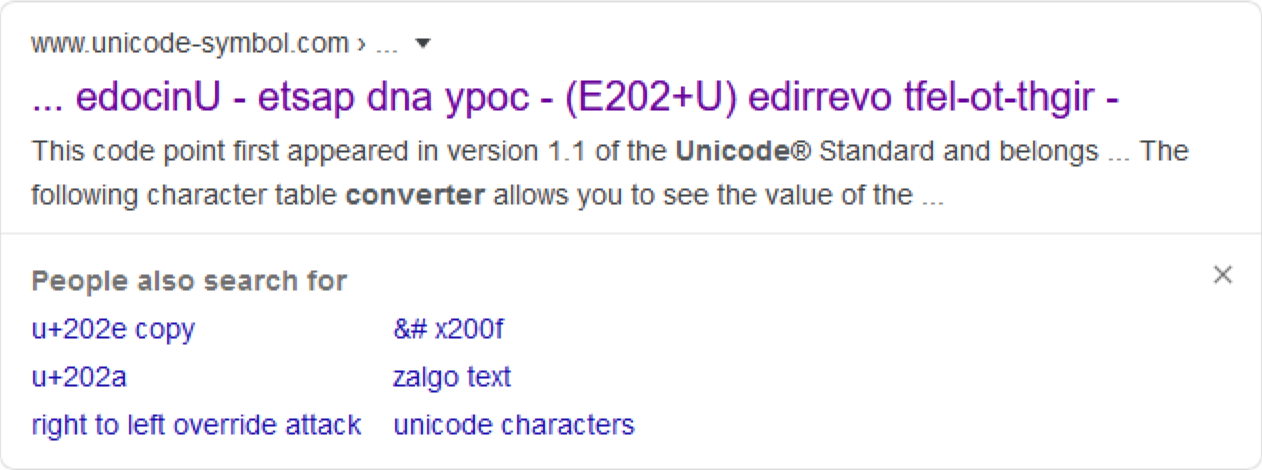
Sanitise text input

- How would you do it?

Sanitise text input

- 𝕯𝖎𝖋𝖋𝖎𝖈𝖚𝖑𝖙 𝖕𝖗𝖔𝖇𝖑𝖊𝖒.

Difficult problem.



www.unicode-symbol.com › ... ▾

... edocinU - etsap dna ypoc - (E202+U) edirrevo tfel-ot-thgir -

This code point first appeared in version 1.1 of the **Unicode**® Standard and belongs ... The following character table **converter** allows you to see the value of the ...

People also search for

u+202e copy

u+202a

right to left override attack

&# x200f

zalgo text

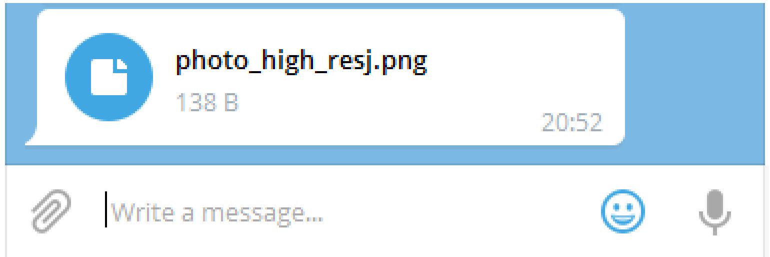
unicode characters

Sanitise text input

- “Unicode injection”: RTL, combining characters, wide characters
- `﷽` is one (1!) character
  `U+FDFD ARABIC LIGATURE BISMILLAH AR-RAHMAN AR-RAHEEM`
- Z̤̱̞̞͎̥Ȧ͎̣͔̞L̥̻̗̳̻͢G̖̯͓̞̣̦̚O̹̹̺!̣̞͇͎̞̬
- 25 different whitespace characters
- Non-printing characters

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

Unicode control characters



photo_high_resj.png

138 B

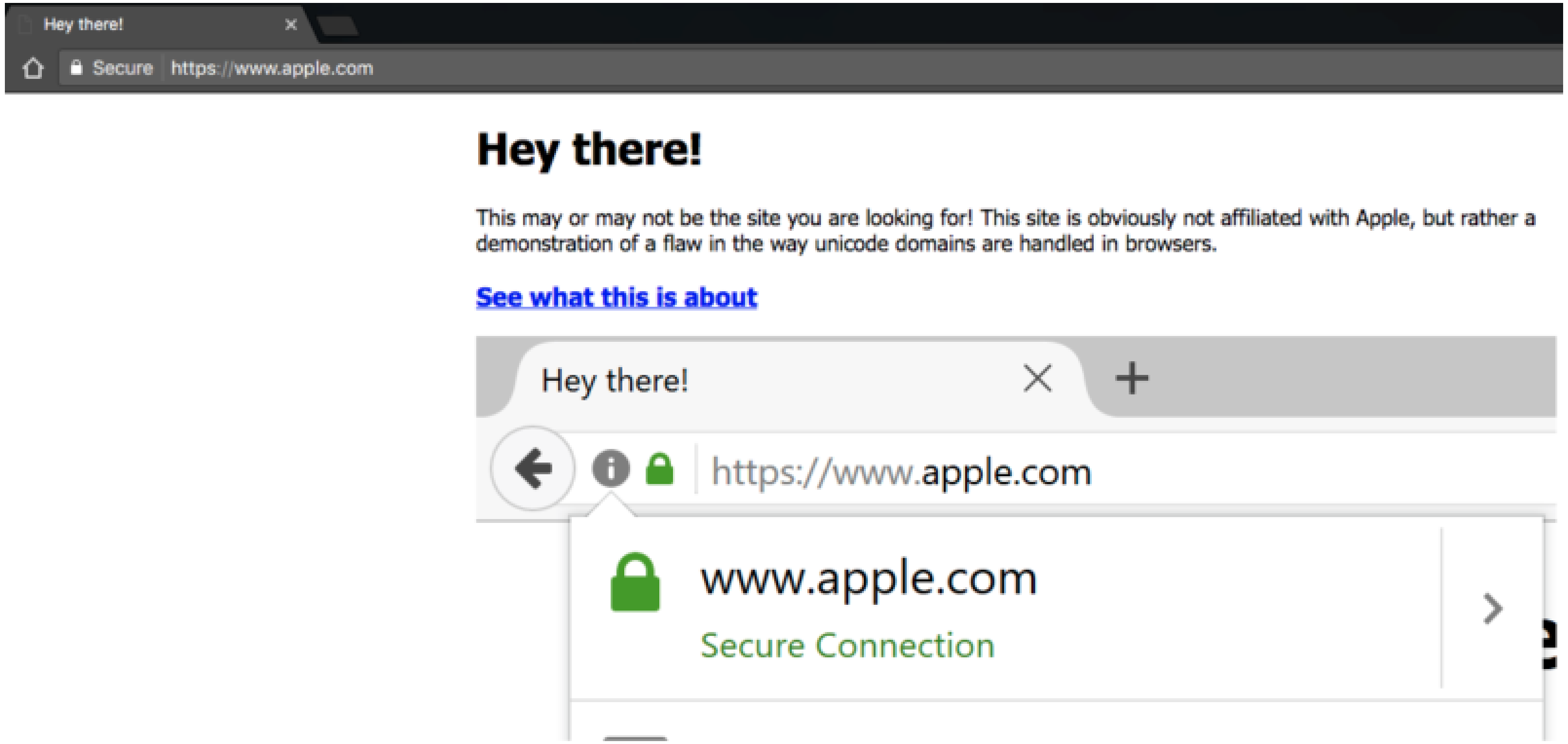
20:52

`photo_high_re` + `U+202E 'RIGHT-TO-LEFT OVERRIDE'` + `gnp.js`

Unicode in URLs

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

Unicode in URLs



Hey there!

This may or may not be the site you are looking for! This site is obviously not affiliated with Apple, but rather a demonstration of a flaw in the way unicode domains are handled in browsers.

See what this is about

Unicode in URLs

`https://www.аррӏе.com/`

`а`	`U+0430 CYRILLIC SMALL LETTER A`
`р`	`U+0440 CYRILLIC SMALL LETTER ER`
`ӏ`	`U+04CF CYRILLIC SMALL LETTER PALOCHKA`
`е`	`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 🍕!
```

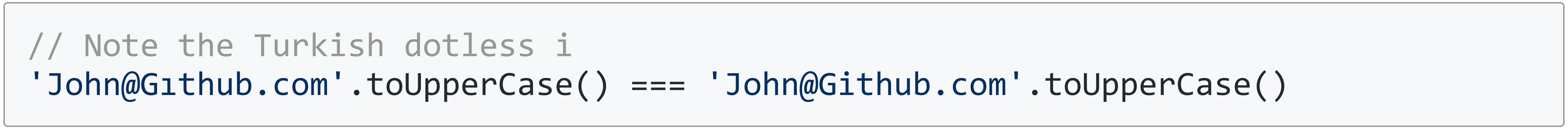
🍕 `sg⁄free.pizza` 🍕

Unicode in URLs

Solution: Use Punycode where/when it makes sense to

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

> Company: GitHub
>
> Vulnerability: Password reset emails delıvered to the wrong address.
>
> Cause: Forgot password emails validated against lowercase value on file, but sent the provided email.



```
// Note the Turkish dotless i
'John@Gıthub.com'.toUpperCase() === 'John@Github.com'.toUpperCase()
```

https://eng.getwisdom.io/hacking-github-with-unicode-dotless-i/

> GitHub's forgot password feature could be compromised because the system lowercased the provided email address and compared it to the email address stored in the user database.

Be careful when normalising or transforming unique identifiers!

Python 3

```
>>> "John@Gıthub.com".lower()
'john@g\xc4\xb1thub.com'
```

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
  ```

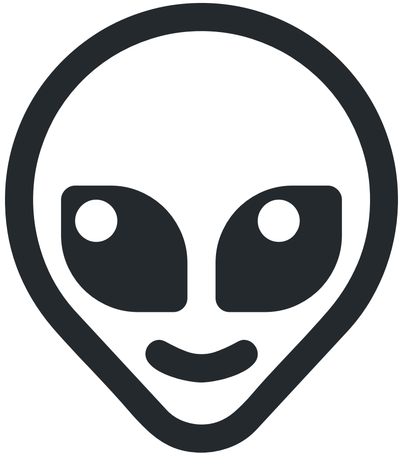
Ill-formed sequences and encoding mismatches

- 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



4 bytes long! `0xF0 0x9F 0x91 0xBD`

Ill-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
```

```
>> 'ﷺ'.length
1

>> 'ﷺ'.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：＼ｗｉｎｄｏｗｓ` 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

```
> 1 + 1;
← 2

> 1 + 1&#894;
       👆
← 🚨 SyntaxError: illegal character 🚨
```

`; U+037E GREEK QUESTION MARK`

Rust shilling

```
error: unknown start of token: \u{37e}
 --> src/lib.rs:1:14
  |
1 | let x = 1 + 1;
  |              ^
  |
help: Unicode character ';' (Greek Question Mark) looks like ';' (Semicolon), but it is not
  |
1 | let x = 1 + 1;
  |              ^                  ^
```

A list of similar characters

Resources

- The Unicode Standard (latest)
- 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