Ɇ₦₵ØĐł₦₲ ₳₦Đ ĐɆ₵ØĐł₦₲ ₣ØⱤ ₦Ɇ₮₩ØⱤ₭ ₮ɆӾ₮ ₮Ɽ₳₦₴ɱł₴₴łØ₦

Overview

- History of computer/network text encoding
  - ASCII
  - Limitations of the ASCII character set
- Unicode
  - Incorporation of all known human character sets
  - Code points
- Unicode Encoding and Decoding
  - UTF-8, UTF-16, UTF-32, etc.
- Unicode examples and Python 3 support

Early Computer Data Transmission

- Created for plain English letter/number focus
- Each letter, number, capital letters, etc., were encoded by assigned binary codes
- American Standard Code for Information Interchange (ASCII)
- Since early computer data transmission was often to peripherals, ASCII also included control character assignments

ASCII

- American Standards Association (now American National Standards Institute, (ANSI)) standard starting in 1960 and first published in 1963 (last update 1986).
- 7-bit codewords in 0-127 range, e.g., “T” : 84; “t” : 116; space: 32.
- 33 non-printing control characters. These affect how text and spaces are processed.
- Now properly referred to as US-ASCII.

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*
1	1	1	1	*[START OF HEADING]*
2	2	10	2	*[START OF TEXT]*
3	3	11	3	*[END OF TEXT]*
4	4	100	4	*[END OF TRANSMISSION]*
5	5	101	5	*[ENQUIRY]*
6	6	110	6	*[ACKNOWLEDGE]*
7	7	111	7	*[BELL]*
8	8	1000	10	*[BACKSPACE]*
9	9	1001	11	*[HORIZONTAL TAB]*
10	A	1010	12	*[LINE FEED]*
11	B	1011	13	*[VERTICAL TAB]*
12	C	1100	14	*[FORM FEED]*
13	D	1101	15	*[CARRIAGE RETURN]*
14	E	1110	16	*[SHIFT OUT]*
15	F	1111	17	*[SHIFT IN]*
16	10	10000	20	*[DATA LINK ESCAPE]*
17	11	10001	21	*[DEVICE CONTROL 1]*
18	12	10010	22	*[DEVICE CONTROL 2]*
19	13	10011	23	*[DEVICE CONTROL 3]*
20	14	10100	24	*[DEVICE CONTROL 4]*
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*
22	16	10110	26	*[SYNCHRONOUS IDLE]*
23	17	10111	27	*[ENG OF TRANS. BLOCK]*
24	18	11000	30	*[CANCEL]*
25	19	11001	31	*[END OF MEDIUM]*
26	1A	11010	32	*[SUBSTITUTE]*
27	1B	11011	33	*[ESCAPE]*
28	1C	11100	34	*[FILE SEPARATOR]*
29	1D	11101	35	*[GROUP SEPARATOR]*
30	1E	11110	36	*[RECORD SEPARATOR]*
31	1F	11111	37	*[UNIT SEPARATOR]*
32	20	100000	40	*[SPACE]*
33	21	100001	41	**!**
34	22	100010	42	**"**
35	23	100011	43	**#**
36	24	100100	44	**$**
37	25	100101	45	**%**
38	26	100110	46	**&**
39	27	100111	47	**'**
40	28	101000	50	**(**
41	29	101001	51	**)**
42	2A	101010	52	*****
43	2B	101011	53	**+**
44	2C	101100	54	**,**
45	2D	101101	55	**-**
46	2E	101110	56	**.**
47	2F	101111	57	**/**
48	30	110000	60	**0**
49	31	110001	61	**1**
50	32	110010	62	**2**
51	33	110011	63	**3**
52	34	110100	64	**4**
53	35	110101	65	**5**
54	36	110110	66	**6**
55	37	110111	67	**7**
56	38	111000	70	**8**
57	39	111001	71	**9**
58	3A	111010	72	**:**
59	3B	111011	73	**;**
60	3C	111100	74	**<**
61	3D	111101	75	**=**
62	3E	111110	76	**>**
63	3F	111111	77	**?**
64	40	1000000	100	**@**
65	41	1000001	101	**A**
66	42	1000010	102	**B**
67	43	1000011	103	**C**
68	44	1000100	104	**D**
69	45	1000101	105	**E**
70	46	1000110	106	**F**
71	47	1000111	107	**G**
72	48	1001000	110	**H**
73	49	1001001	111	**I**
74	4A	1001010	112	**J**
75	4B	1001011	113	**K**
76	4C	1001100	114	**L**
77	4D	1001101	115	**M**
78	4E	1001110	116	**N**
79	4F	1001111	117	**O**
80	50	1010000	120	**P**
81	51	1010001	121	**Q**
82	52	1010010	122	**R**
83	53	1010011	123	**S**
84	54	1010100	124	**T**
85	55	1010101	125	**U**
86	56	1010110	126	**V**
87	57	1010111	127	**W**
88	58	1011000	130	**X**
89	59	1011001	131	**Y**
90	5A	1011010	132	**Z**
91	5B	1011011	133	**[**
92	5C	1011100	134	****
93	5D	1011101	135	**]**
94	5E	1011110	136	**^**
95	5F	1011111	137	**_**
96	60	1100000	140	**`**
97	61	1100001	141	**a**
98	62	1100010	142	**b**
99	63	1100011	143	**c**
100	64	1100100	144	**d**
101	65	1100101	145	**e**
102	66	1100110	146	**f**
103	67	1100111	147	**g**
104	68	1101000	150	**h**
105	69	1101001	151	**i**
106	6A	1101010	152	**j**
107	6B	1101011	153	**k**
108	6C	1101100	154	**l**
109	6D	1101101	155	**m**
110	6E	1101110	156	**n**
111	6F	1101111	157	**o**
112	70	1110000	160	**p**
113	71	1110001	161	**q**
114	72	1110010	162	**r**
115	73	1110011	163	**s**
116	74	1110100	164	**t**
117	75	1110101	165	**u**
118	76	1110110	166	**v**
119	77	1110111	167	**w**
120	78	1111000	170	**x**
121	79	1111001	171	**y**
122	7A	1111010	172	**z**
123	7B	1111011	173	**{**
124	7C	1111100	174	**|**
125	7D	1111101	175	**}**
126	7E	1111110	176	**~**
127	7F	1111111	177	*[DEL]*

What about the 8th Bit?

- Many computers were based on 8-bit data words (bytes)
- ASCII left half of the available codes undefined i.e., 128-255.
- Sometimes used as a parity bit
- The spare codewords started to be assigned in non-standard ways, e.g., some industry proprietary, some assigned in non-English speaking countries. This created an interoperability mess.

Text Encoding Example

- Extended Binary Coded Decimal Interchange Code (EBCDIC) devised by IBM (1963/1964) for peripheral communications. 8-bit code.
- EDBDIC defined different "code pages" to extend encoding to accommodate different languages.
- A code page is a table of values that describes the character set used for encoding
- Various "double byte" character sets were used to encode (mainly) Eastern languages.
- There are many other text encodings, e.g.,

ISO-8859-1 (Latin-1)

- Upward compatible ASCII extension that uses all 8 bits and adds 96 additional characters. Many Western European languages are completely covered:
- Afrikaans, Albanian, Basque, Breton, Corsican, Danish[a], English, Faroese, Galician, German, Icelandic, Irish, Indonesian, Italian, Kurdish, Leonese, Luxembourgish, Malay, Manx, Norwegian, Occitan, Portuguese, Rhaeto-Romanic, Scottish Gaelic, Spanish, Swahili, Swedish, Walloon
- About 7% of web sites used this encoding in 2016.
- ISO-8859-1 is the first 256 code points in Unicode

Other Language Encodings

- Windows-1252 adds 27 more characters
- Latin-2, Latin-3, ..., Latin-9, (i.e., ISO-8859-2, ISO-8859-3,..., ISO-8859-9) target other (mainly Western European) languages.
- Many other languages are missing!
- How to accommodate all world-wide languages?

The Unicode Standard

Unicode

- Intended to be the final world-wide standard text representation mechanism, e.g., enables standard world-wide data communications.
- All the major living scripts are included.
- Defines "abstract characters" for all languages., i.e., basic units of textual information.
- Gives a name and a unique binary "code point" identifier to each abstract character.
- Code points are not intended for data communication.

Emojis have been recently added, e.g.,

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams
1	U+1F600								
2	U+1F603								
3	U+1F604								
4	U+1F601								
5	U+1F606								

For a complete list, go to:
https://unicode.org/emoji/charts/full-emoji-list.html

Unicode Example

- Character: M
- Name: LATIN CAPITAL LETTER M
- Categories: uppercase letter, Left-to-Right
- Codepoint: U+004D
- i.e., "M" (Latin Capital Letter M) is assigned unicode code point of 004D, written as U+004D, i.e., a 16 bit word.
- Legal Unicode code point values: U+0000 to U+10FFFF (over 1 million characters). Only about 10% have have been assigned.

Unicode

- Prior approaches involve assigning a given character to a unique binary word that is used for transmission, e.g., ASCII assigns "a" to 97, i.e., transmitted as 01100001.
- Unicode instead assigns each symbol in an alphabet to a unique binary "code point". Code points are not intended for communication.

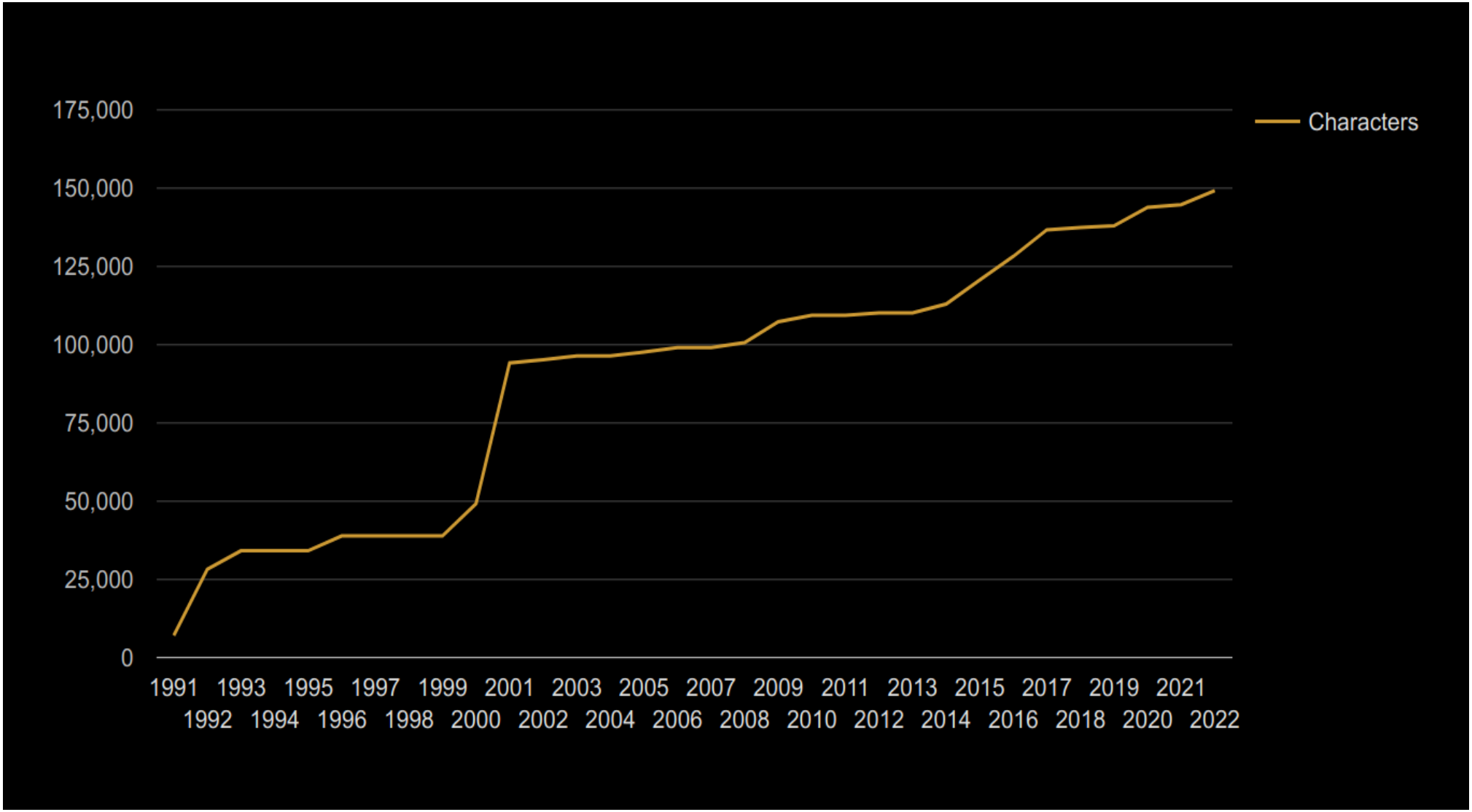
Unicode

- Example: "M" (Latin Capital Letter M) is assigned unicode code point of 004D, written as U+004D, i.e., a 16 bit word.
- Code points are multi-byte words. To transmit them, we would need to be careful about byte ordering, e.g., The word "cat" consists of the unicode sequence: U+0063 U+0061 U+0074
- Big-endian: 0063 0061 0074
- Little-endian: 0074 0061 0063
- Would probably need to include a byte order mark (BOM) in transmitted data.

Unicode

- Latest version of Unicode contains 149,813 characters that cover over 135 modern and historical scripts, as well as multiple symbol sets.
- Rules for things such as text normalization, composition, decomposition, collation, and bidirectional display order.
- Latest version is Unicode 15.1.0, maintained by the Unicode Consortium.

Unicode Codepoint Growth



Unicode Character Map

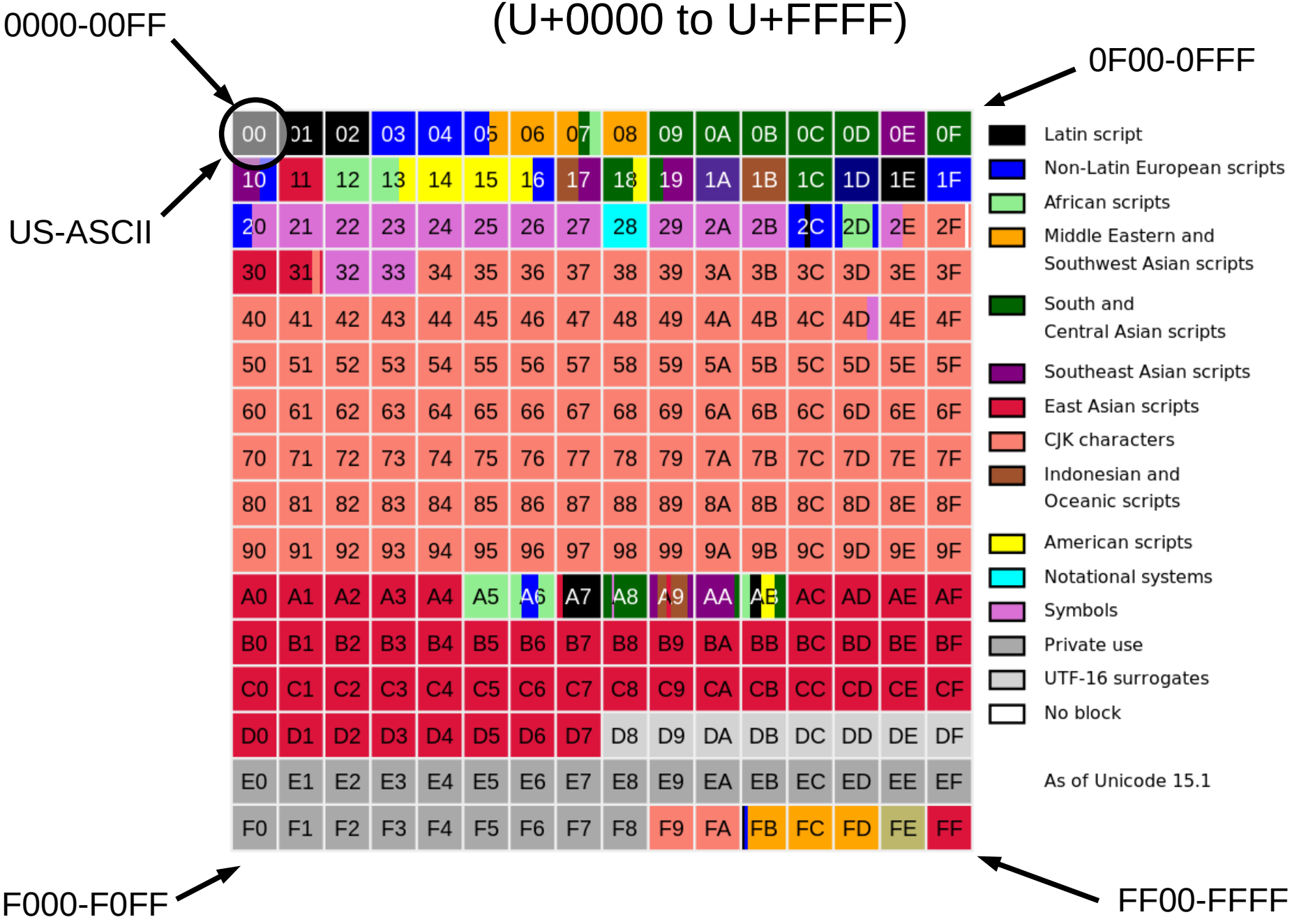
- Contains code planes of 64K (FFFF) code points each. Planes currently defined:
  - 0000 : Basic Multilingual Plane (BMP)
  - 0001 : Supplementary Multilingual Plane (SMP)
  - 0002 : Supplementary Ideographic Plane (SIP)
  - 0003 : Tertiary Ideographic Plane (TIP)
  - 000E: Supplementary Special-purpose Plane (SSP)
  - 000F: Private Use Plane (PUP)
  - 0010 : Private Use Plane (PUP)

Unicode Code Plane Geometry

- Each small block: $2^8 = 256$ code points
- Each row: 16 (2^4) small blocks = $2^4 \times 2^8 = 2^{12}$ = 4K (4096) code points
- Each plane: 16 (2^4) rows = $2^4 \times 2^{12} = 2^{16}$ = 64K (65536) code points

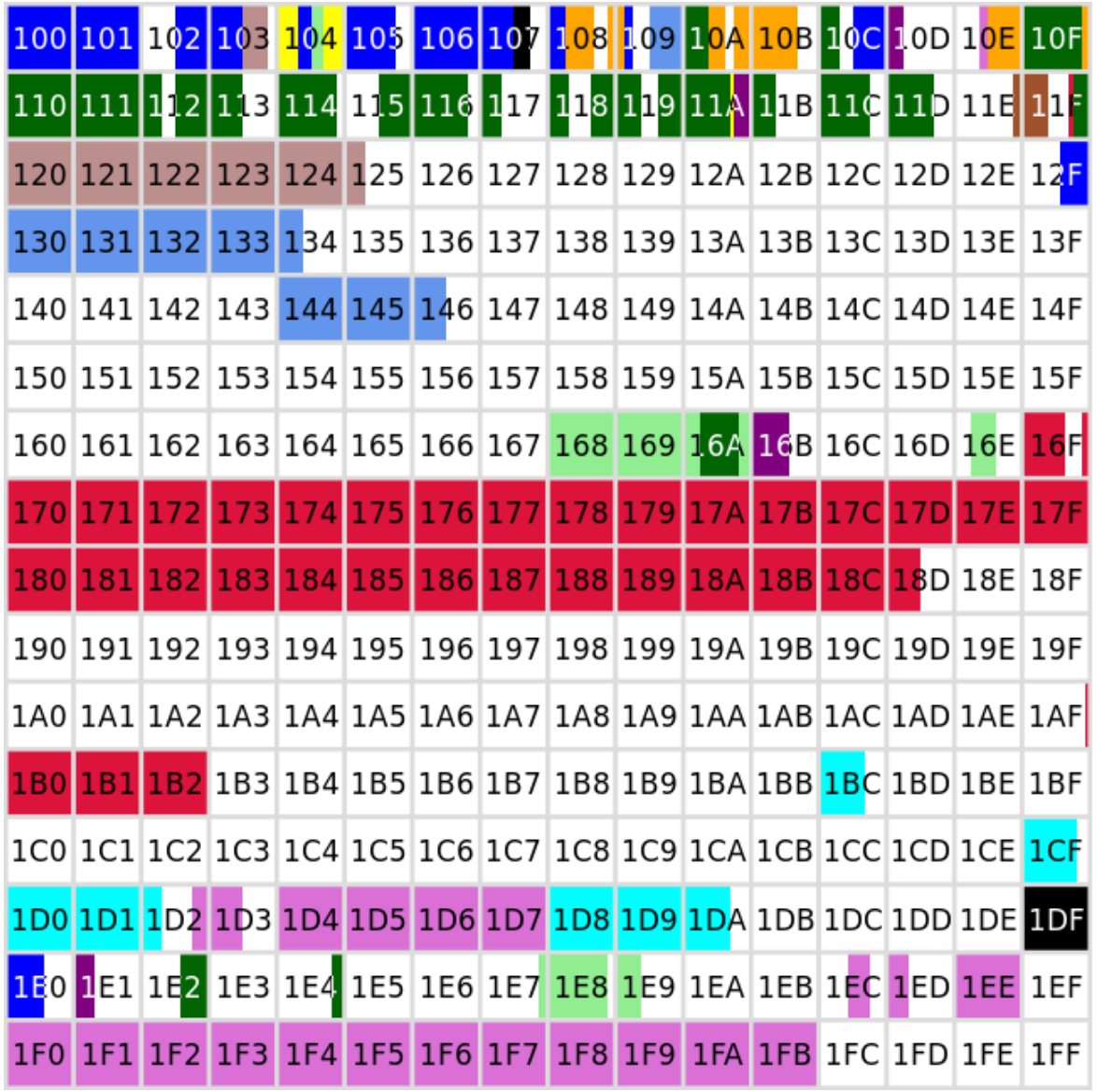
Plane 0: Basic Multilingual

(U+0000 to U+FFFF)



Plane 1: Supplementary Multilingual

(U+10000 to U+1FFFF)

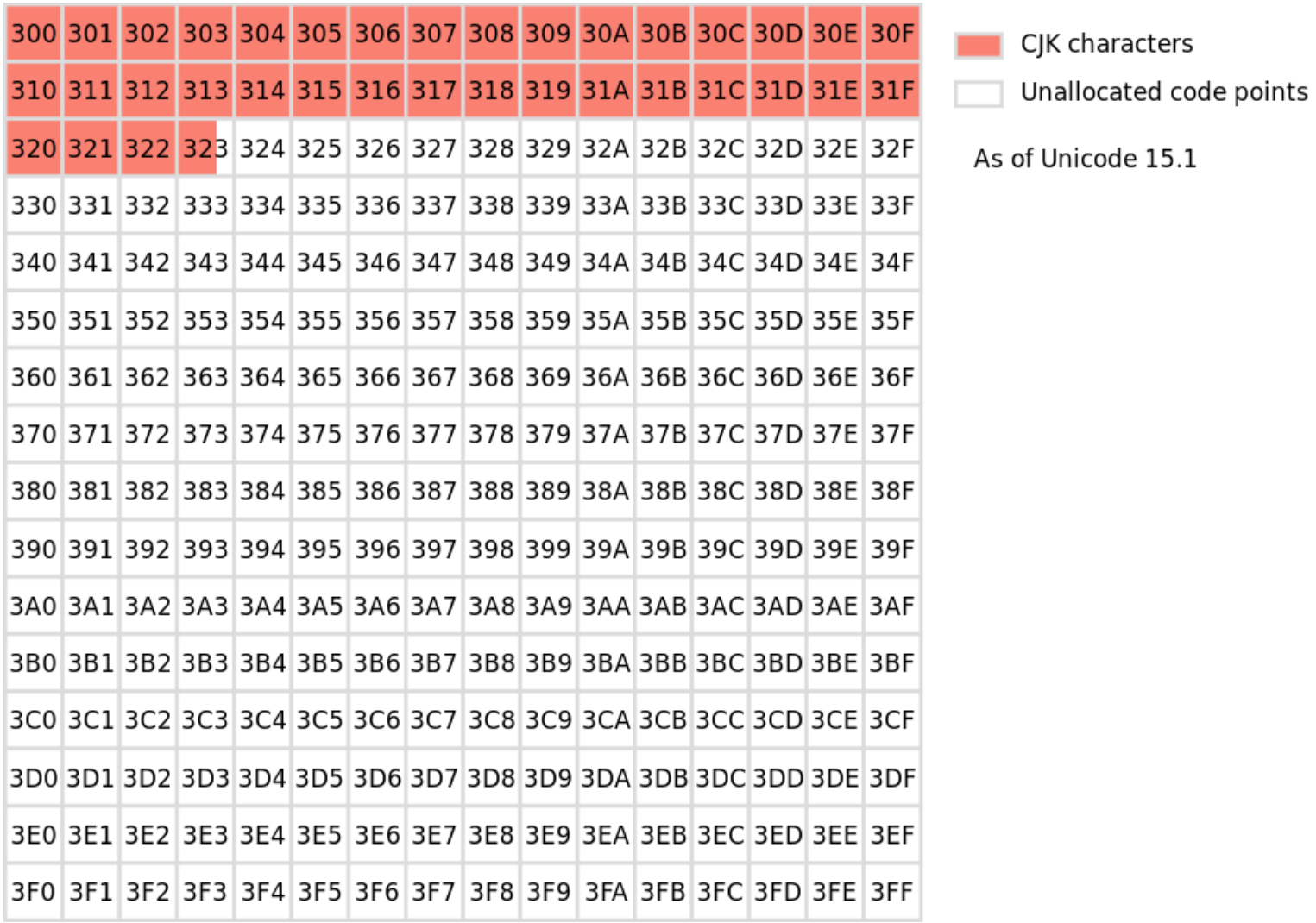


Plane 2: Supplementary Ideographic

(U+20000 to U+2FFFF)

Plane 3: Tertiary Ideographic

(U+30000 to U+3FFFF)

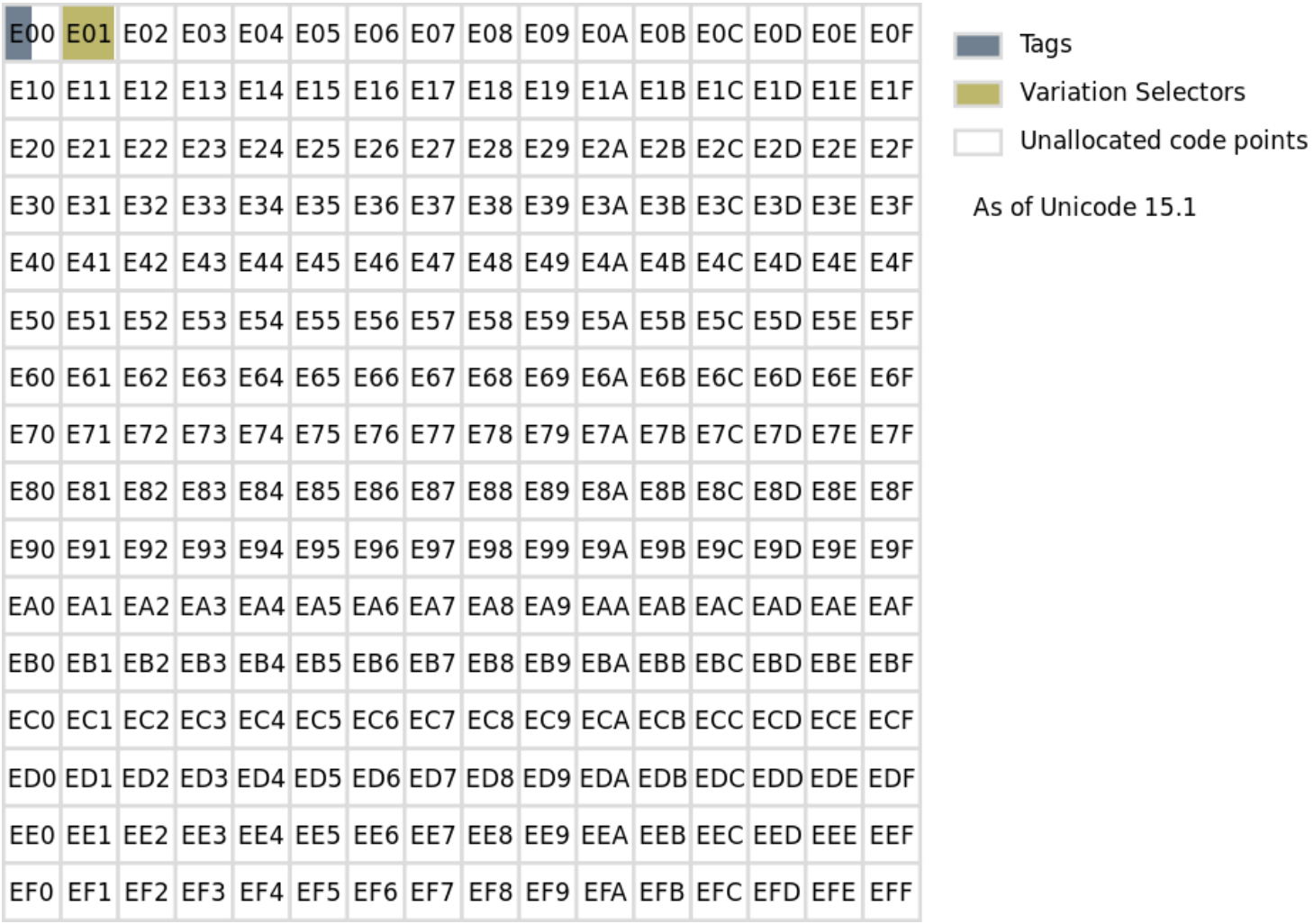


Planes 4 to 13: Unused

(U+40000 to U+DFFFF)

Plane 14: Supplementary Special-purpose

(U+E0000 to U+EFFFF)



Planes 15/16: Supplementary Private Use

(U+F0000 to U+10FFFF)

To see the Unicode character definitions, go to:

http://unicode.org/charts/

Unicode

Unicode Support

Привет мир
Γειά σου Κόσμε
你好，世界
Bonjour le monde
Hallo Welt
Helló Világ
שלום עולם
مرحبا بالعالم
こんにちは世界
Ahoj světe
Selam Dünya

Unicode Encodings

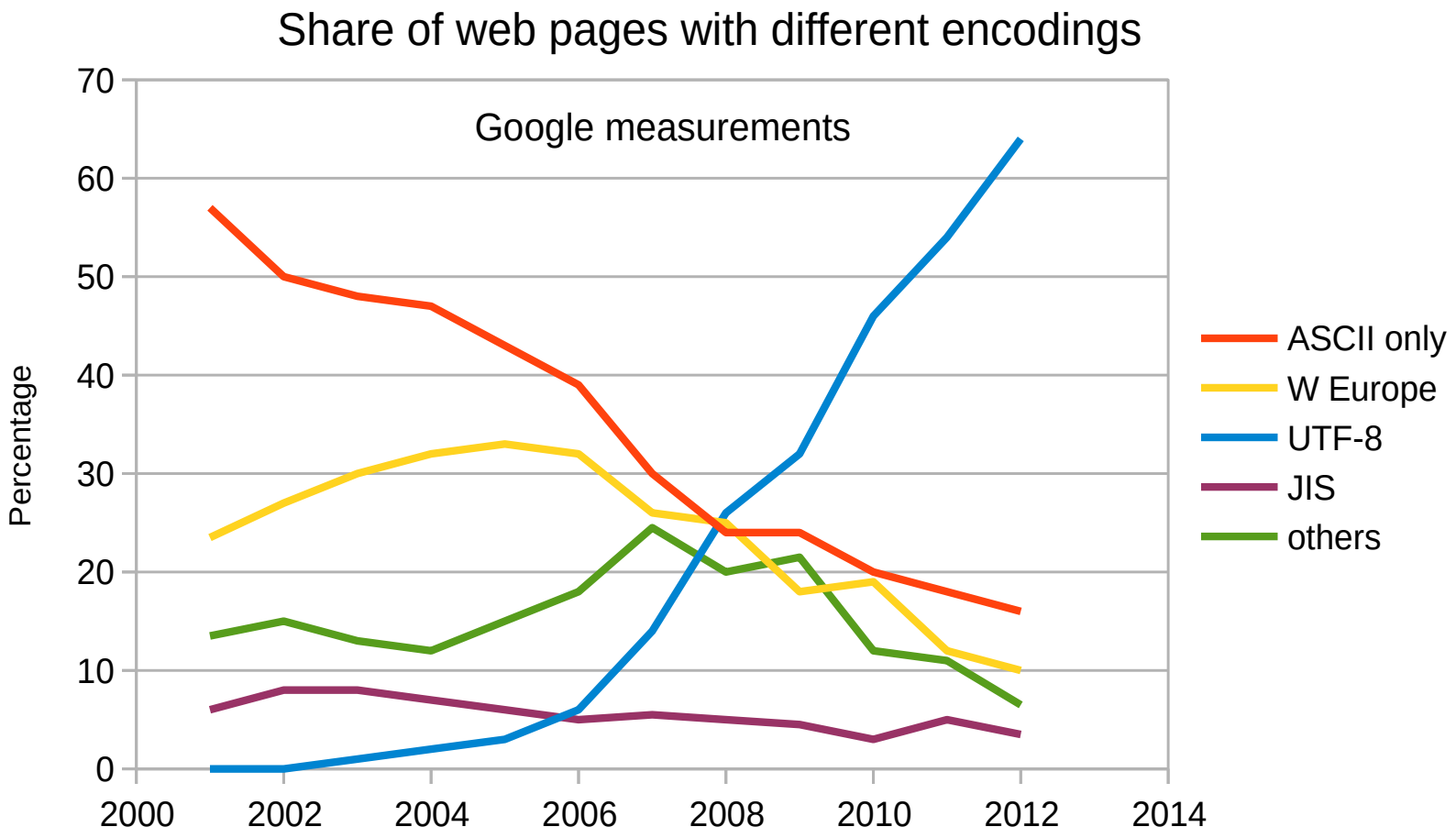
Unicode Encodings

- Instead of transmitting unicode code points, unicode defines different "encodings" that can be applied prior to transmission.
- Unicode Transformation Format (UTF)
  - UTF defines mappings of Unicode code points to a unique byte sequence.
  - The UTF mappings are reversible in that a given character is uniquely defined by both its unicode code point and its UTF encodings.
  - Most text in documents and webpages is encoded using some of the various UTF encodings

Unicode Encodings

- Unicode Transformation Format (UTF)
  - The conversions between all UTF encodings are algorithmically based, fast and lossless
  - Makes it easy to support data input or output in multiple formats, while using a particular UTF for internal storage or processing
  - Common encodings:
    - UTF-8
    - UTF-16
    - UTF-32.

Web Page Encoding Usage



As of 2024, over 98% of web pages use UTF-8 encoding. The second is ISO-8859-1 with about 1.3%.

UTF-8 Encoding

	Code Point	Byte 1	Byte 2	Byte 3	Byte 4
7 bits	0000-007F	0xxxxxxx			
11 bits	0080-07FF	110xxxxx	10xxxxxx		
16 bits	0800-D7FF	1110xxxx		10xxxxxx	
16 bits	E000-FFFF				
21 bits	10000-10FFFF	11110xxx			10xxxxxx

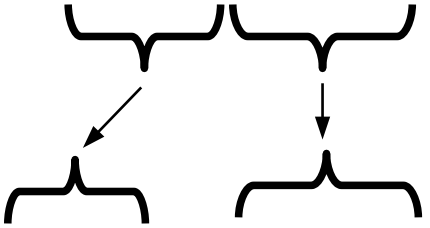
- Variable length (1 to 4 bytes). Sent as a sequence of bytes, i.e., no byte order issues.
- Note that the number of leading ones tells the number of bytes in the UTF-8 encoding (except for the 1-byte case).
- "continuation bytes" all start with 10.

UTF-8 Encoding

	Code Point	Byte 1	Byte 2	Byte 3	Byte 4
7 bits	0000-007F	0xxxxxxx			
11 bits	0080-07FF	110xxxxx	10xxxxxx		
16 bits	0800-D7FF	1110xxxx		10xxxxxx	
16 bits	E000-FFFF				
21 bits	10000-10FFFF	11110xxx			10xxxxxx

- Example: Ω : Greek Capital Letter Omega

  Unicode: U+03A9 (0000 0011 1010 1001)



  UTF-8: 110 01110 10 101001 (CEA9 hex)

- Note that Unicode code points for US-ASCII characters is their US-ASCII encoding and also their UTF-8 encoding.

UTF-16 Encoding

	Code Point	16 bit word 1	16 bit word 2
16 bits	0000-D7FF	xxxxxxxxxxxxxxxx	
16 bits	E000-FFFF		
20 bits	10000-10FFFF	110110xxxxxxxxxx	110111xxxxxxxxxx

- Variable length (1 or 2 16-bit words). Sent as a sequence of sixteen bit words, i.e., byte order issues, i.e., UTF-16BE and UTF-16LE.
- Encoding uses gaps in code point assignment:

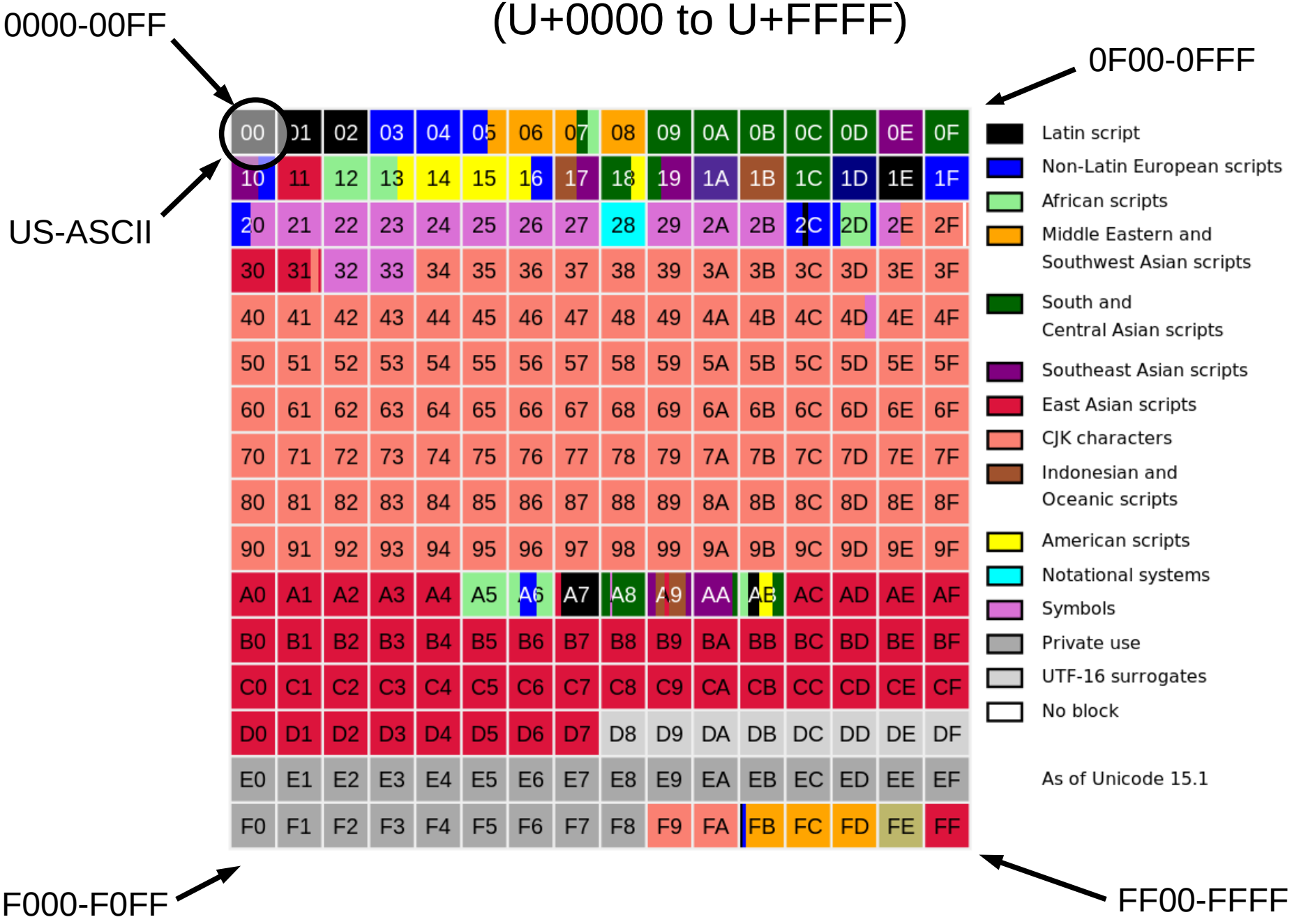
  110110xxxxxxxxxx = D800 - DBFF

  110111xxxxxxxxxx = DC00 – DFFF

  i.e., code points in the above ranges would start with 110110 or 110111 which interfere with the 6-bit UTF-16 preambles so those code points have been reserved, never to be assigned a Unicode character.

Plane 0: Basic Multilingual

(U+0000 to U+FFFF)



UTF-32 Encoding

	Code Point	32 bit word
21 bits	0000-D7FF	00000000000xxxxxxxxxxxxxxxxxxxx
21 bits	E000-10FFFF	

- Fixed length (one 32 bit word). Need to deal with byte order, i.e., UTF-32BE and UTF-32LE.
- Very simple but lots of overhead.
- Its main use is in internal APIs where the data is single code points or glyphs, rather than strings of characters.

Python 3 Unicode Support

- Python 3 has full support for Unicode.
- `str` type is stored as Unicode characters. Unicode chars can be entered as literals using name or code point, e.g.,

  ```
  d = "\N{GREEK CAPITAL LETTER OMEGA}"

  e = "\u0394"
  ```

- `chr(i)` returns the string representing a character whose Unicode code point is the integer `i`. Valid range for the argument is from 0 through 1,114,111 (0x10FFFF hex).

  See chr_examples.py.

- `ord(c)`: Given a string representing one Unicode character, return an integer representing the Unicode code point of that character.

  `ord(c)` and `chr(i)` are inverses.

  See ord_examples.py

Python3 Unicode Support

- Encode/decode functions convert between unicode and unicode encodings, e.g.,
- `str.encode()` : returns a bytes object representation of the Unicode string, encoded in the requested encoding.
- `bytes.decode()` : returns a Unicode string, decoded using the requested decoding.
- In our network transmissions of text we encode prior to transmission and decode after reception.

Python Unicode Support

- Python3 encode/decode functions convert between unicode and unicode encodings, e.g.,

```
str_1 = "hello"

str_1_utf8 = str_1.encode('utf-8')
```

- This creates a "bytes" object with UTF-8 encoding. To decode we can

```
str_1_decode = str_1.decode('utf-8')
```

- See encodedecode_examples.py.
- See print_encodings.py