Information, Characters, Unicode

Hidden Moral

Small mistakes can be catastrophic!

Style
Care about every character of your program.

Tip: printf
Care about every character in the program's output.

(Be reasonably tolerant and defensive about the input. "Fail early" and clearly.)

Imperative

Thou ſhalt care about every character in your program.

Corollaries

Thou ſhalt know every character
in the input.

Thou ſhalt care about every character
in your output.

Information – Characters

In modern computing, natural-language text is very important information. (“number-crunching” is less important.) Characters of text are represented in several different ways and a known character encoding is necessary to exchange text information.

For many years an important encoding standard for characters has been US ASCII–a 7-bit encoding. Since 7 does not divide 32, the ubiquitous word size of computers, 8-bit encodings are more common. Very common is ISO 8859-1 aka “Latin-1,” and other 8-bit encodings of characters sets for languages other than English.

Currently, a very large multi-lingual character repertoire known as Unicode is important.

Information – Characters

Bits are not information until the relevant parties agree and what they represent. A standard is required to successfully communicate a character of text. The bits are mostly arbitrary choices.

binary	oct	dec	hex	char	
0110 0001	041	97	0x61	a	THE LETTER 'A'
0110 0010	042	98	0x62	b	THE LETTER 'B'
0110 0011	043	99	0x63	c	THE LETTER 'C'

Blocks of n bits have 2^n different bit patterns and so 2^n characters can be represented.

ASCII (American Standard Code for Information Interchange), is a 7-bit character encoding standard for digital communication. It has defined $2^7 = 128$ bit patterns.

It was one of the first standards for encoding symbols (letters, numbers, and punctuation used in English text). This fixed-width encoding evolved in the 1960s by the institution for standards for the United States. It has been in widespread use for information exchange ever since, but now supplanted by other standards. A survey (2023) suggests that US-ASCII is used by far less than 1% of websites and UTF-8 (described later) by 98% of websites (`w3techs.com`). (But UTF-8 retains US-ASCII.)

The Internet Assigned Numbers Authority (IANA) prefers the name US-ASCII for this character encoding.

Some US-ASCII Characters

Each character has a unique bit pattern used to represent it (and a Unicode name as we shall see later).

binary	oct	dec	char	Unicode	
0000 1001	0011	9	HT	U+0009	HORIZONTAL TABULATION
0010 0000	0040	32		U+0020	SPACE
0010 1110	0056	46	.	U+002E	FULL STOP
0010 1111	0057	47	/	U+002F	SOLIDUS
0011 0000	0060	48	0	U+0030	DIGIT ZERO
0011 0001	0061	49	1	U+0031	DIGIT ONE

Although 8 bits are shown above, only 7 bits are used in the US-ASCII standard.

US ASCII (7-bit), or bottom half of Latin1

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SS	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETP	CAN	EM	SUB	ESC	FS	GS	RS	US
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Compromises

With only 2^7 bit pattern (really 2^5) many compromises were made. Some characters did double duty.

binary	oct	dec	char	Unicode	
0010 0111	0047	39	'	U+0027	APOSTROPHE
0010 1010	0052	42	*	U+002A	ASTERISK
0010 1101	0055	45	-	U+002D	HYPHEN-MINUS

There is a discretization problem when left and right single quotation marks are represented by the US-ASCII bit patterns for the apostrophe and grave accent characters.

Alice in Wonderland

```
It was all very well to say "Drink me," but the wise little Alice was
not going to do _that_ in a hurry.  "No, I’ll look first," she said,
"and see whether it’s marked ’_poison_’ or not";
```

It was all very well to say “Drink me,” but the wise little Alice was
not going to do *that* in a hurry. “No, I’ll look first,” she said,
“and see whether it’s marked ‘*poison*’ or not”;

Combining Characters

Diacritic marks (uncommon in English) are parts of characters. The design of US-ASCII includes diacritic marks and this gives rise to the notion of combining characters in encode letters like ô or è.

binary	oct	dec	char	Unicode	
0101 1110	0136	94	^	U+005E	CICUMFLEX ACCENT
0110 0000	0140	96	`	U+0060	GRAVE ACCENT

Control Characters

Notice that the first twos rows are filled with so-called control characters. These characters have no printable representation and were introduced to control hardware. For example: ␇ "ring the bell." Except for various conventions for indicating lines of text, most of these characters have no use today. So, nearly one-quarter of the space available for representing characters is wasted.

Of course, the space character does not have a printable representation (no ink is used to print a space), but it is extremely useful.

Top half of Latin-1

With 8 bits 256 characters can be encoded. Latin-1 is twice as big as US-ASCII. The extra characters allow languages like Icelandic, Spanish and German to be written in Latin-1.

XXX	XXX	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
DCS	PU1	PU2	SST	CCH	MS	SPA	EPA	SOS	XXX	SCI	CSI	ST	OSC	PM	APC
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Some Characters

Here are some of the characters in Latin-1 not used in writing English.

binary	oct	dec		Unicode	
1100 0011	0303	195	Ã	U+00C3	LATIN CAPITAL LETTER A WITH TILDE
1101 0111	0327	215	×	U+00D7	MULTIPLICATION SIGN
1101 1111	0337	223	ß	U+00DF	LATIN SMALL LETTER SHARP S
1110 1101	0355	237	í	U+00ED	LATIN SMALL LETTER I WITH ACUTE
1111 1110	0376	254	þ	U+00FE	LATIN SMALL LETTER THORN

An 8-bit character set is a convenient size and so US-ASCII is for the most part replaced by Latin-1 which supports some European languages. Microsoft's CP1252 is somewhat similar.

Some Characters

For fear of low-level bit confusion the two rows of control characters were repeated in the section with the 8th bit set.

binary	oct	dec		Unicode	
`1001 0001`	`0231`	145	PU1	`U+0091`	*private use one*
`1001 0010`	`0232`	146	PU2	`U+0092`	*private use two*
`1001 1000`	`0240`	152	SOS	`U+0098`	*start of string*
`1001 1011`	`0243`	155	CSI	`U+009B`	*control sequence introducer*

(So-called "private use" code points were introduced in the C1 controls. These are reserved for private parties to agree upon.)

The new ISO 8859-15 (Latin-9) nicknamed Latin-0 updates Latin-1 by replacing eight infrequently used characters ¤¦¨´¸¼½¾ with left-out French letters (ÿ, œ) and Finnish and Lithuanian letters (š, ž), and placing the Euro sign € in the cell 0xA4 of the former (unspecified) currency sign ¤.

¤	U+00A4	CURRENCY SIGN	$\longrightarrow$ €	U+20AC	EURO SIGN
¦	U+00A6	BROKEN BAR	$\longrightarrow$ Š	U+0160	LATIN CAPITAL LETTER S WITH CARON
¨	U+00A8	DIAERESIS	$\longrightarrow$ š	U+0161	LATIN SMALL LETTER S WITH CARON
´	U+00B4	ACUTE ACCENT	$\longrightarrow$ Ž	U+017D	LATIN CAPITAL LETTER Z WITH CARON
¸	U+00B8	CEDILLA	$\longrightarrow$ ž	U+017E	LATIN SMALL LETTER Z WITH CARON
¼	U+00BC	VULGAR FRAC 1 QUARTER	$\longrightarrow$ Œ	U+0152	LATIN CAPITAL LIGATURE OE
½	U+00BD	VULGAR FRACTION 1 HALF	$\longrightarrow$ œ	U+0153	LATIN SMALL LIGATURE OE
¾	U+00BE	VULGAR FRAC 3 QUARTERS	$\longrightarrow$ Ÿ	U+0178	LATIN CAPITAL LETTER Y WITH DIAERESIS

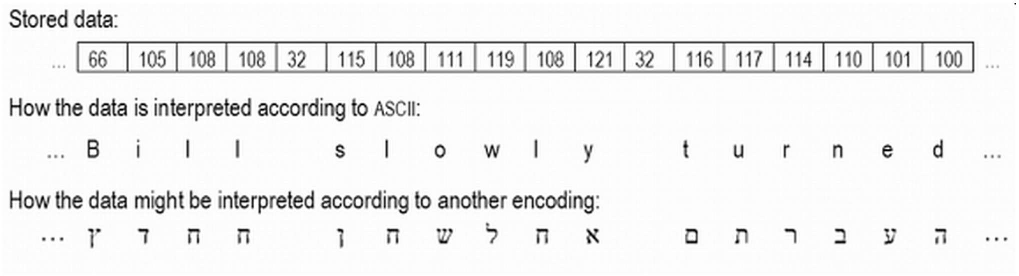
Top half of Latin-0

PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGC	SCI	CSI	ST	OSC	PM	APC
	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
đ	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Differences in Character Encodings

binary	oct	dec	hex	MacR	1252	Latin1	Latin0
0111 0011	0163	115	0x73	s	s	s	s
1000 0000	0200	128	0x80	Ä	€	XXX	PAD
1000 0101	0205	135	0x85	Ö	. . .	NEL	NEL
1000 1010	0212	138	0x8A	ä	Š	VTS	VTS
1010 0100	0244	164	0xA4	§	¤	¤	€
1010 0110	0246	166	0xA6	¶	¦	¦	Š
1011 0110	0266	182	0xB6	∂	¶	¶	¶
1101 1011	0333	219	0xDB	€	Û	Û	Û
1110 0100	0344	228	0xE4	‰	ä	ä	ä
1111 0011	0363	243	0xF3	Û	ó	ó	ó

Standards help insure that the bit patterns are understood the same way. But which standard? The applicable standard must be clearly known.



Stored data:

...	66	105	108	108	32	115	108	111	119	108	121	32	116	117	114	110	101	100	...

How the data is interpreted according to ASCII:

...	B	i	l	l		s	l	o	w	l	y		t	u	r	n	e	d	...

How the data might be interpreted according to another encoding:

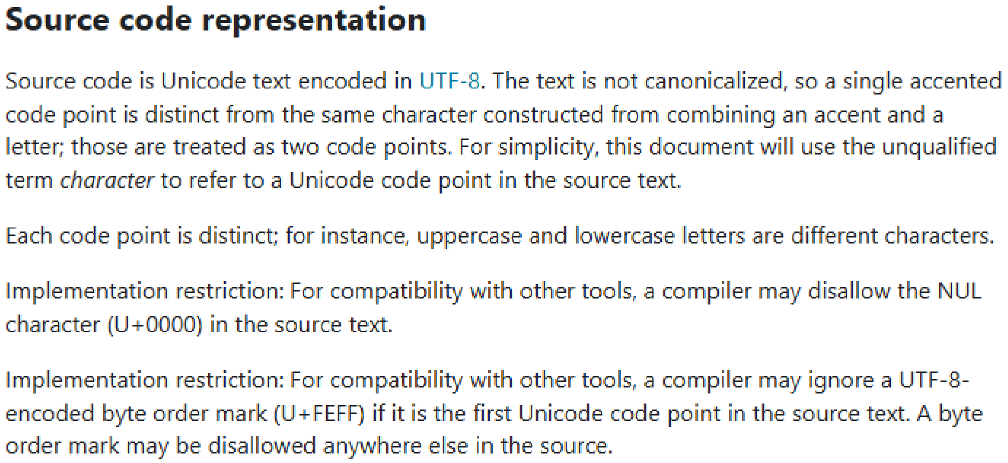
...	ץ	ד	ח	ח		ן	ח	ש	ל	ח	א		ם	ת	ר	ב	ע	ה	...

Source Code

Source code is a text file

What encoding does it use? Ada, Go, Haskell, Java, Python use Unicode.

Go Source Code



Source code representation

Source code is Unicode text encoded in UTF-8. The text is not canonicalized, so a single accented code point is distinct from the same character constructed from combining an accent and a letter; those are treated as two code points. For simplicity, this document will use the unqualified term *character* to refer to a Unicode code point in the source text.

Each code point is distinct; for instance, uppercase and lowercase letters are different characters.

Implementation restriction: For compatibility with other tools, a compiler may disallow the NUL character (U+0000) in the source text.

Implementation restriction: For compatibility with other tools, a compiler may ignore a UTF-8-encoded byte order mark (U+FEFF) if it is the first Unicode code point in the source text. A byte order mark may be disallowed anywhere else in the source.

Haskell Source Code

"Haskell uses the Unicode character set. However, source programs are currently biased toward the ASCII character set used in earlier versions of Haskell.

This syntax depends on properties of the Unicode characters as defined by the Unicode consortium. Haskell compilers are expected to make use of new versions of Unicode as they are made available."

Python Source Code

Python supports writing source code in UTF-8 by default, but you can use almost any encoding if you declare the encoding being used. This is done by including a special comment as either the first or second line of the source file Python looks for `coding: name` or `coding=name` in the comment.

```
# encoding: latin-1  [Must be on 1st or 2nd line]

u = 'abcdé'
print(ord(u[-1]))
```

If you don't include such a comment, the default encoding used will be UTF-8 as already mentioned. See PEP 263 for more information.

Python Tip

```
    # -*- coding: ascii -*-
```

Emacs recognizes the character set name "ascii" and variables in the three character sequence `-*-` See Emacs §22.18 Charsets and Python recognizes the character set name "ascii" See Python known Standard Encodings.
Of, if you are not using Emacs:

```
# This Python file uses the following encoding: ascii
```

Java Source Code

You can write a Java program in any character set. Because Unicode is essentially a super set of all characters sets the programs is first translated into Unicode.

```
javac                      SourceCode.java # platform default
javac -encoding US-ASCII SourceCode.java
javac -encoding ASCII      SourceCode.java # ASCII = US-ASCII
javac -encoding UTF-8      SourceCode.java # What about BOM?
javac -encoding UTF8       SourceCode.java # UTF-8 = UTF8
javac -encoding utf8       SourceCode.java # UTF-8 = utf8
javac -encoding latin1     SourceCode.java
javac -encoding latin0     SourceCode.java
javac -encoding utf32      SourceCode.java
javac -encoding cp1252     SourceCode.java
```

The encoding UTF-8 in Java 11 does not like the optional byte order mark (BOM). (But then neither do I.) This may have been a bug.

Encoding of Data Stream

Indicate to the `Scanner` class which character encoding is to be expected, and Java will interpret the bytes correctly. This is because Java uses Unicode internally which is a super-set of all commonly used character set encodings.

```java
Scanner s = new Scanner (System.in, "LATIN-1");
```

```java
Scanner s = new Scanner (System.in, "Cp1252");
```

Without a specified character encoding, the computer's default encoding is used.

```java
Scanner s = new Scanner (System.in);
```

A program with such a scanner may behave differently on different computers.

Java Tip

All the input in lab will be US-ASCII, so please use the two argument form of the `Scanner` class at all times.

```java
Scanner s = new Scanner (System.in, "US-ASCII");
```

A bad bit pattern in the input (there won't be one) would result in a Unicode character for "bad character" – 0xFFFD REPLACEMENT CHARACTER – in your input. The behavior of the programs for the exercises in lab is never defined on any "bad" input whatsoever. The program may do anything at all including loop or result in a runtime error. In particular the program does not have to detect "bad" input or report it.

Java Tip

As of Java 18 IO is done in the default character set regardless of the platform. The default character set `java.nio.charset.Charset.defaultCharset()` is UTF-8, *unless* overridden by the system property `file.encoding`.

It is no longer necessary to be explicit about the character set in order to be platform independent, but it does not bad practice to be explicit anyway.

One can specify the platform dependent character set with the value `COMPAT` for `file.encoding`. The platform characters set typically depends upon the locale and charset of the underlying operating system.

Whitespace

Six invisible or white-space characters are legal in a Java program. No other control characters are legal in a Java program. A Java program used to be permitted to end with the “substitute” character.

binary	dec	Latin1	Unicode	
0000 1001	9	HT	U+0009	HORIZONTAL TABULATION
0000 1010	10	LF	U+000A	LINE FEED
0000 1100	12	FF	U+000C	FORM FEED
0000 1101	13	CR	U+000D	CARRIAGE RETURN
0001 1010	26	SUB	U+001A	SUBSTITUTE
0010 0000	32		U+0020	SPACE

There is no advantage to using a horizontal tabulation or a substitute character in a Java program. But there is a risk of breaking some application that uses Java source code for input (pretty-printers, text beautifiers, metric tools, etc.) Newlines indications are necessary for formatting programs, and Java permits all three of the common newline conventions: the line feed character (common in Unix applications), the carriage return (Mac applications), and the carriage return character followed by the line feed (Microsoft applications).

MacOS	CR	"\r"
Unix	LF	"\n"
Windows	CR,LF	"\r\n"

Other newline markers are much less common. Next-line (NEL x85) is used to mark end-of-line on some IBM mainframes. Unicode has its own approach to indicating a new line:

Unicode	
U+2028	LINE SEPARATOR

The method readLine() of the Java class BufferedReader tolerates MacOS, Unix or Window (but not Unicode) new-line indications.

The Java JVM initializes the new-line indication according to the platform it is running. This value is available through system properties: System.getProperty("line.separator"). The method format in class Formatter recognizes %n in format strings as the value of the line separator.

```
// Use Unicode line separator character
System.setProperty("line.separator","\u2028");
System.out.format("A line.%n");

# Python
print ("A line.", end=u"\2028")
print ("A line.", end="\r\n")
print ("A line.", end="\n")
print ("A line.")  # end="\n" by default
```

Newline

From Wikipedia:

> *In computing a newline, also known as a line break or end-of-line (EOL) marker, is a special character or sequence of characters signifying the end of a line of text.*
>
> *There is also some confusion whether newlines terminate or separate lines. If a newline is considered a separator, there will be no newline after the last line of a file. The general convention on most systems is to add a newline even after the last line, i.e. to treat newline as a line terminator. Some programs have problems processing the last line of a file if it is not newline terminated.*

Thou ſhalt end every line
in your program and output
with a line terminator.

Conveniently, the number of line terminators in a file is the number of lines in the file.

Failing to abide by this convention may lead to miscommunication and the loss of points on tests and programs.

What Do Characters Mean?

Some characters, like the tab, have no fixed meaning, even though it has an agreed upon code point. Tabs are interpreted differently by different applications leading to confusion.

binary	dec	Latin1	Unicode	
0000 1001	9	HT	U+0009	HORIZONTAL TABULATION
1010 0100	164	¤	U+00A4	CURRENCY SIGN

What good is a standard when the meaning is unclear?

A consortium of companies got together in the 1990's to solve the information confusion caused by competing character encodings by creating one universal encoding for everybody.

The consortium thought at first it would be a simple, fixed-width (16 bit) encoding. Java, developed at the same time, immediately adopted the standard instead of defining their own units for Java source and Java strings.

Unicode Versions

version	date	scripts	characters	bits
1.0	Oct 1991	24	7,161	12.81
2.0	Jul 1996	24	38,950	15.25
3.0	Sep 1999	38	49,249	15.59
4.0	Apr 2003	52	96,447	16.56
5.0	Jul 2006	64	99,098	16.60
6.0	Oct 2010	93	109,449	16.74
7.0	Jun 2014	123	113,021	16.79
8.0	Jun 2015	123	120,737	16.88
9.0	Jun 2016	135	128,237	16.97

Here characters mean the number of encoded characters: graphic characters, format characters plus control characters. This does not include unassigned code points (permanently reserved), private use characters, non-characters (66 code points), or surrogate code points (2,048 reserved for the convenience of multi-byte encoding).

Unicode Versions

version	date	scripts	characters	bits
6.0	Oct 2010	93	109,449	16.74
7.0	Jun 2014	123	113,021	16.79
8.0	Jun 2015	123	120,737	16.88
9.0	Jun 2016	135	128,237	16.97
10.0	Jun 2017	139	136,690	17.06
11.0	Jun 2018	146	137,374	17.07
12.0	Mar 2019	150	137,928	17.07
13.0	Mar 2020	154	143,849	17.13
14.0	Sep 2021	159	144,697	17.14
15.0	Sep 2022	161	149,186	17.19

Unicode 13.0 (2020 March 10) adds 5,930 characters, for a total of 143,859 (wiki: +65=143,924) characters, and the first block in plane 3 for CJK ideographs. These additions include 4 new scripts, for a total of 154 scripts, as well as 55 new emoji characters. Characters: segmented digits. Scripts: historic Yezidi, historic Chorasmian

Unicode 14.0 (2021 Sept 14) adds 838 characters, for a total of 144,697. These additions include 5 new scripts, for a total of 159 scripts, as well as 37 new emoji characters.

Unicode 15.0 (2022 Sept 13) adds 4,489 characters, for a total of 149,186. These additions include 2 new scripts (historical Kawi and Mundari), for a total of 161 scripts, as well as 20 new emoji characters, and 4,192 CJK ideographs.

The character counts above exclude code points with General Category CC (ASCII control characters). There are exactly 65 such code points and this will never change. Wikipedia includes graphic, format and control characters in the character count, and excludes private-use characters, non-characters, and surrogate code points (used for encoding).

Some Unicode Characters

binary		Unicode	
0000 0000 1111 0110	ö	U+00F6	LATIN SMALL LETTER O WITH DIAERESIS
0000 0001 0100 0010	ł	U+0142	LATIN SMALL LETTER L WITH STROKE
0000 0001 0111 0101	ŵ	U+0175	LATIN SMALL LETTER W WITH CIRCUMFLEX
0000 0011 1001 0100	Δ	U+0394	GREEK CAPITAL LETTER DELTA
0010 0000 0010 1000		U+2028	LINE SEPARATOR
0010 0010 0010 1011	$\int$	U+222B	INTEGRAL
0010 0010 0100 0111	$\ncong$	U+2247	NEITHER APPROXIMATELY NOR ACTUALLY EQUAL TO
0010 0010 1001 0111	$\otimes$	U+2297	CIRCLED TIMES

The customary way to refer to a Unicode point is with `U+` followed by the 4 (or 5) hexadecimal digits.

Where do characters come from?

A recent graphic character added to Unicode is a currency symbol for the Turkish Lira. It is the winning design of a competition held by the Turkish central bank. The design for the symbol was revealed to the public in March 2012. It was the only character added to Unicode 6.2 in September 2012. The bitcoin symbol (designed by the creator of the currency) was added in Unicode 10; the Som sign (Kyrgyzstan) in Unicode 14.

₺ ₿ ⃀

ə ɲ ʕ ɠ IPA

℃ ℆ ℔ ℘ ℞ Letter-like

∄ ∑ ∰ ≔ ⊵ Mathematical

✂ ✈ ✎ ✔ ✘ Dingbats

丄 字 文 直 迎 骨 Ideographs

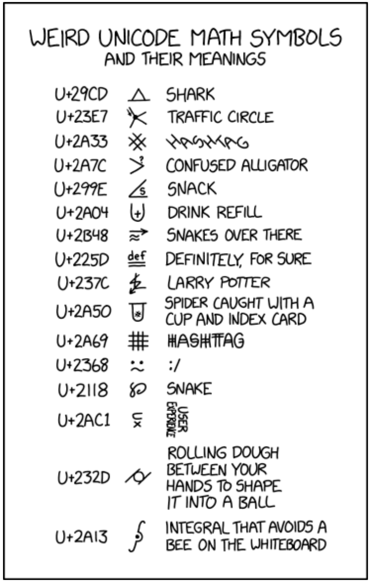
Other Symbols

Multiple snowflake symbols are encoded in Unicode including: “snowflake" at U+2744; “tight trifoliate snowflake" at U+2745; and “heavy chevron snowflake" at U+2746.

❄ ❅ ❆

Multiple hand symbols are encoded in Unicode including: “reversed victory hand” at U+1F594, “reversed hand with middle finger extended” at U+1F595, “raised hand with part between middle and ring fingers” at U+1F596.

🖓 🖔 🖕 🖖 🖗


WEIRD UNICODE MATH SYMBOLS
AND THEIR MEANINGS
U+29CD SHARK
U+23E7 TRAFFIC CIRCLE
U+2A33
U+2A7C CONFUSED ALLIGATOR
U+299E SNACK
U+2A04 DRINK REFILL
U+2B48 SNAKES OVER THERE
U+225D DEFINITELY, FOR SURE
U+237C LARRY POTTER
U+2A50 SPIDER CAUGHT WITH A CUP AND INDEX CARD
U+2A69 HASHTAG
U+2368 :/
U+2118 SNAKE
U+2AC1 USER EXPERIENCE
U+232D ROLLING DOUGH BETWEEN YOUR HANDS TO SHAPE IT INTO A BALL
U+2A13 INTEGRAL THAT AVOIDS A BEE ON THE WHITEBOARD

xkcd

`https://xkcd.com/2606/`

Making fun of all the Unicode math symbols

Roll-over pop-up: U+2A0B Mathematicians need to calm down

Unicode Mistakes

In something as complex as encoding all the writing systems of the world, mistakes are bound to happen.

	Unicode	
★	U+156F	CANADIAN SYLLABICS TTH
☺	U+263A	WHITE SMILING FACE

Meaning of U+156F is said by Unicode to be “probably a mistaken interpretation of an asterisk used to mark a proper noun”

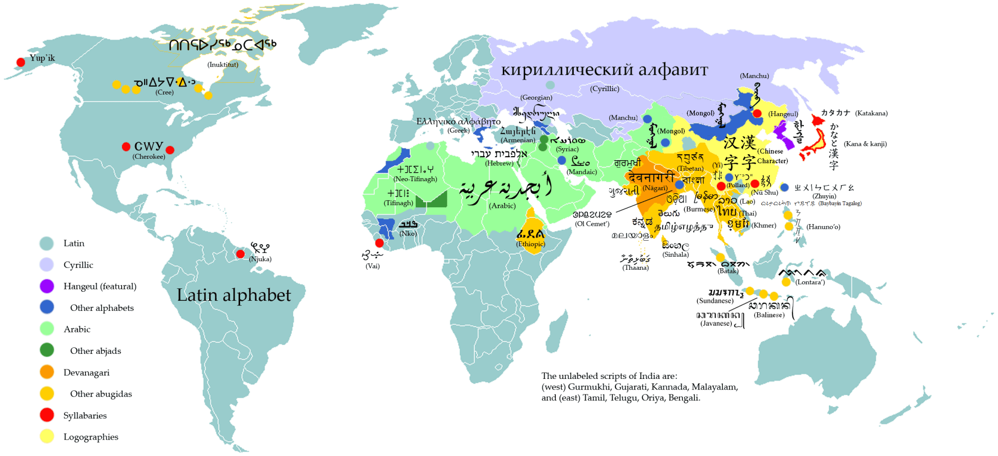
U+263A predates emoji.

Unicode Mistakes

See the code points marked "correction" in the text file `NameAliases.txt`.
There are enough code points so that a few can be wasted.

The consortium works hard to vet proposals for inclusion, but it is gigantic semiotic undertaking.

Unicode has now been around long enough that the gradual changes in written communications of society are a factor.

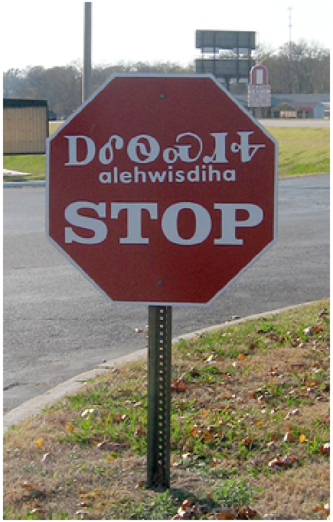


The unlabeled scripts of India are:
(west) Gurmukhi, Gujarati, Kannada, Malayalam,
and (east) Tamil, Telugu, Oriya, Bengali.

Unicode Supports Many Scripts

Latin:	A	B	C	D	E	...		
Arabic:	...	خ	ح	ج	ث	ت	ب	ا
Hebrew:	...	ז	ו	ה	ד	ג	ב	א
Armenian:	Ա	Բ	Գ	Դ	Ե	...		
Cyrillic:	А	Б	В	Г	Д	...		
Devanagari:	क	ख	ग	घ	ङ	च	छ	...
Thai:	ก	ข	ฃ	ค	ฅ	...		

Cherokee Script



glyph	Unicode	
Ꭰ	U+13A0	CHEROKEE LETTER A
Ꭱ	U+13A1	CHEROKEE LETTER E
Ꭲ	U+13A2	CHEROKEE LETTER I
Ꭳ	U+13A3	CHEROKEE LETTER O
Ꭴ	U+13A4	CHEROKEE LETTER U
Ꭵ	U+13A5	CHEROKEE LETTER V
Ꭷ	U+13A7	CHEROKEE LETTER KA
Ꮞ	U+13CE	CHEROKEE LETTER SE
Ꮢ	U+13D2	CHEROKEE LETTER SV

(Notice the similarity with the Latin letter forms. The Cherokee script was invented by Sequoyah around 1820 with knowledge of the Latin script, but without consideration of English sounds.)

Thou ſhalt not reaſon by analogy.

Unicode Emoji

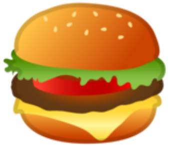
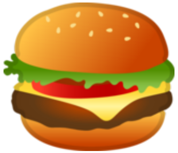
The word emoji comes from the Japanese: (e-mo-ji).

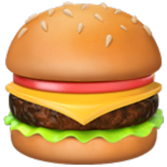
絵	文	字
`U+7D75`	`U+6587`	`U+5B57`
e	mo	ji
picture	writing	character

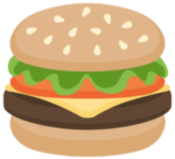
Unicode Emoji

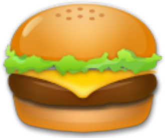
Some emoji as designed by Apple.

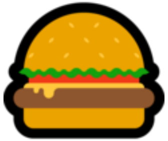
Google “Cheeseburger” [U+1f354] Controversy (2017)

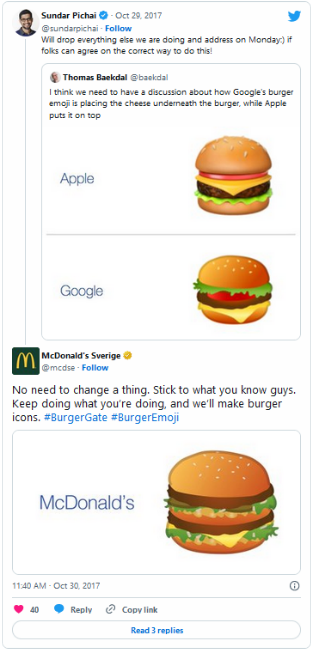




Apple


Facebook


LG5


Microsoft



Sundar Pichai · Oct 29, 2017
@sundarpichai · Follow
Will drop everything else we are doing and address on Monday:) if folks can agree on the correct way to do this!

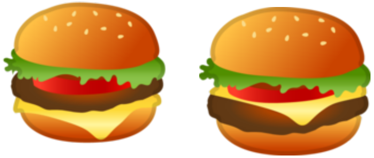
Thomas Baekdal @baekdal
I think we need to have a discussion about how Google's burger emoji is placing the cheese underneath the burger, while Apple puts it on top

McDonald's Sverige
@mcdse · Follow

No need to change a thing. Stick to what you know guys. Keep doing what you're doing, and we'll make burger icons. #BurgerGate #BurgerEmoji

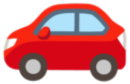
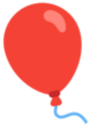
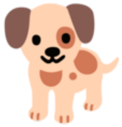
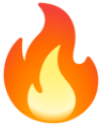
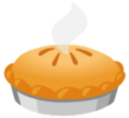
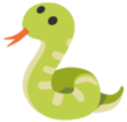
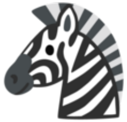
11:40 AM · Oct 30, 2017

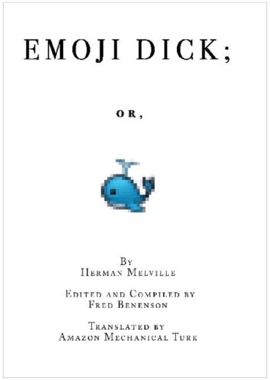
2017 Controversy
Google CEO Sundar Pichai



Google "Cheeseburger" [U+1f354]

Google Unicode Emoji

auto	ballon	dog	fire	pie	snake	zebra
						
U+1F697	U+1F388	U+1F415	U+1F525	U+1F967	U+1F40D	U+1F993



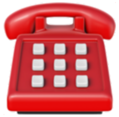
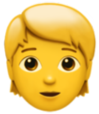
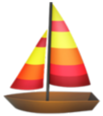
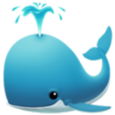
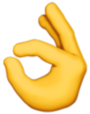
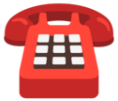
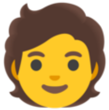
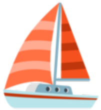
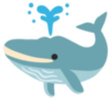
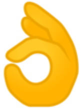
EMOJI DICK;

OR,

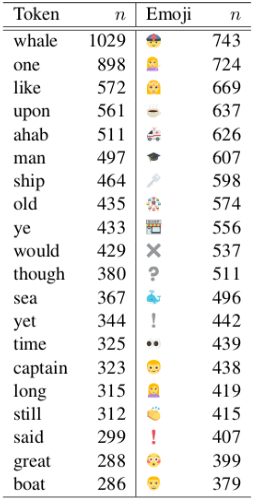
By
Herman Melville

Edited and Compiled by
Fred Benenson

Translated by
Amazon Mechanical Turk

Unicode:	U+260E	U+1F9D1	U+26F5	U+1F433	U+1F4CC
	:telephone:	:person:	:sailboat:	:spouting_whale:	:ok_hand:
Apple:					
Google:					

Call me Ismael.



Token	n	Emoji	n
whale	1029	👲	743
one	898	💁	724
like	572	👸	669
upon	561	☕	637
ahab	511	🚑	626
man	497	🎓	607
ship	464	🔑	598
old	435	🎡	574
ye	433	🏪	556
would	429	✖	537
though	380	❔	511
sea	367	🐳	496
yet	344	❕	442
time	325	👀	439
captain	323	👦	438
long	315	🙎	419
still	312	👏	415
said	299	❗	407
great	288	😳	399
boat	286	👨	379

Table 1: The top 20 tokens and emoji by individual frequency.

"Call me Ishmael" – How do you translate emoji? by Will Radford, Andrew Chisholm, Ben Hachey, Bo Ha

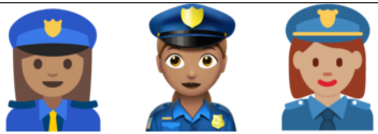
Fitzpatrick Skin Color

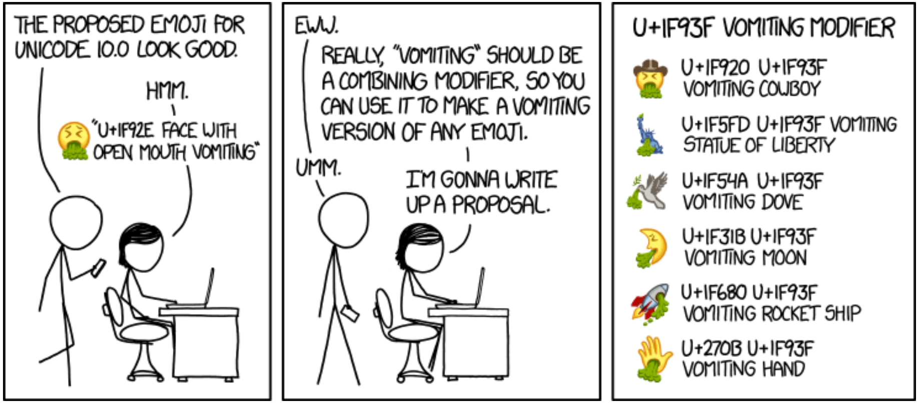
Skin tone modifiers were released as part of Unicode 8.0.

	Unicode	
■	U+1F3FB	EMOJI MODIFIER FITZPATRIC TYPE-1-2
■	U+1F3FC	EMOJI MODIFIER FITZPATRIC TYPE-3
■	U+1F3FD	EMOJI MODIFIER FITZPATRIC TYPE-4
■	U+1F3FD	EMOJI MODIFIER FITZPATRIC TYPE-5
■	U+1F3FD	EMOJI MODIFIER FITZPATRIC TYPE-6

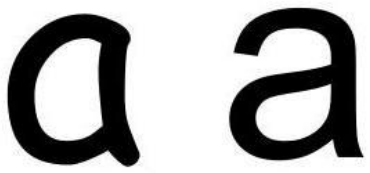
Emoji Sequences

An example sequence: police officer, medium skin color, zero width joiner, female, request emoji presentation

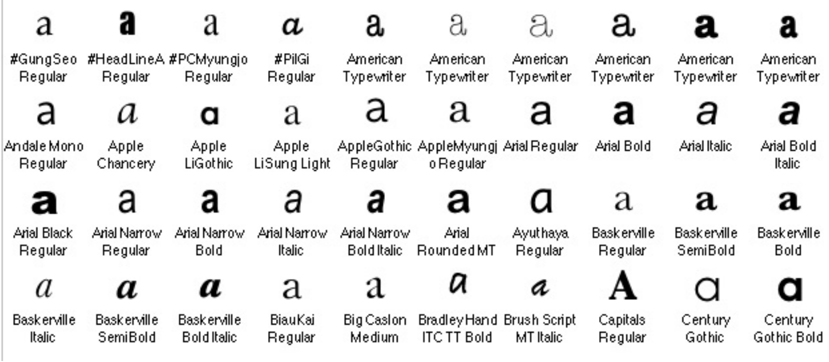
	Unicode	
👮	U+1F46E	POLICE OFFICER
🏽	U+1F3FD	EMOJI MODIFIER FITZPATRIC TYPE-4
	U+ 200D	ZERO WIDTH JOINER
♀	U+ 2640	FEMALE SIGN
	U+ FE0F	VARIATION SELECTOR-16
		

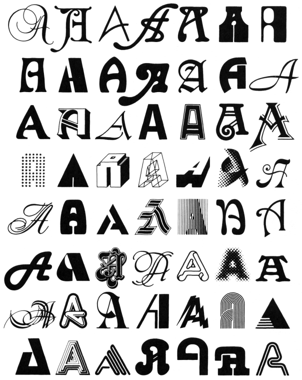

THE PROPOSED EMOJI FOR UNICODE 10.0 LOOK GOOD.
HMM.
"U+1F92E FACE WITH OPEN MOUTH VOMITING"
EWW.
REALLY, "VOMITING" SHOULD BE A COMBINING MODIFIER, SO YOU CAN USE IT TO MAKE A VOMITING VERSION OF ANY EMOJI.
UMM.
I'M GONNA WRITE UP A PROPOSAL.
U+1F93F VOMITING MODIFIER
U+1F920 U+1F93F VOMITING COWBOY
U+1F5FD U+1F93F VOMITING STATUE OF LIBERTY
U+1F54A U+1F93F VOMITING DOVE
U+1F31B U+1F93F VOMITING MOON
U+1F680 U+1F93F VOMITING ROCKET SHIP
U+270B U+1F93F VOMITING HAND

Glyph Versus Character



It is economical to encode the information content of writing regardless of the variety of forms. A character is the unit of information; a glyph is a particular form in which a character is represented. A letter in English may have different forms, but it means the same thing.


#GungSeo Regular
#HeadLineA Regular
#PCMyungjo Regular
#PilGi Regular
American Typewriter
American Typewriter
American Typewriter
American Typewriter
American Typewriter
American Typewriter
Andale Mono Regular
Apple Chancery
Apple LiGothic
Apple LiSung Light
AppleGothic Regular
AppleMyungjo Regular
Arial Regular
Arial Bold
Arial Italic
Arial Bold Italic
Arial Black Regular
Arial Narrow Regular
Arial Narrow Bold
Arial Narrow Italic
Arial Narrow Bold Italic
Arial Rounded MT
Ayuthaya Regular
Baskerville Regular
Baskerville SemiBold
Baskerville Bold
Baskerville Italic
Baskerville SemiBold
Baskerville Bold Italic
BiauKai Regular
Big Caslon Medium
Bradley Hand ITC TT Bold
Brush Script MT Italic
Capitals Regular
Century Gothic
Century Gothic Bold



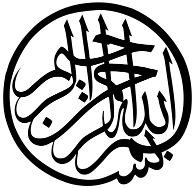
	A	B	C	D	E	F	G
1	Balmoral	Cardinal	Squire	Glastonbury	Arnold Böcklin	Bottleneck	Countdown
2	Eckmann Schrift	Futura Black	Hobo	Lazybones	Old English	Revue	Park Avenue
3	Romic Bold	Tintoretto	Vivaldi	Univers 67	Airkraft	Apollo	Algerian
4	Astra	Baby Teeth	Block Up	Bombere	Buster	Calypso	Columbian Italic
5	Aristocrat	Company	Glaser Stencil	Cathedral	Good Vibrations	Le Golf	Harrington
6	Harlow Solid	Motter Ombra	Masquerade	Phyllis	Pluto Outline	Process	Primitive
7	Magnificat	Quicksilver	Raphael	Roco	Shatter	Stripes	Sinaloa
8	Stop	Stack	Piccadilly	Neptun	Motter Tektura	Odin	Yagi Link Double

As far as possible a code point is assigned an abstract character and not a particular glyph. However, it is not always clear how to discretize the information content of scripts. For example, the Latin, Greek, and Cyrillic capital A (U+0041, U+0391, U+0410) have different code points.

A A A

Yet, the glyphs for these characters are the same.

Latin $A, B, \ldots, R, S, T, U$ versus Greek $A, B, \ldots, P, \Sigma, T, \Upsilon$.
This confusion along with the subtext that goes in the language of mathmatics, required a remark in HMU, page 142, footnote 2: “That ’T’ should be thought of as a Greek captial tau, the letter following sigma.”



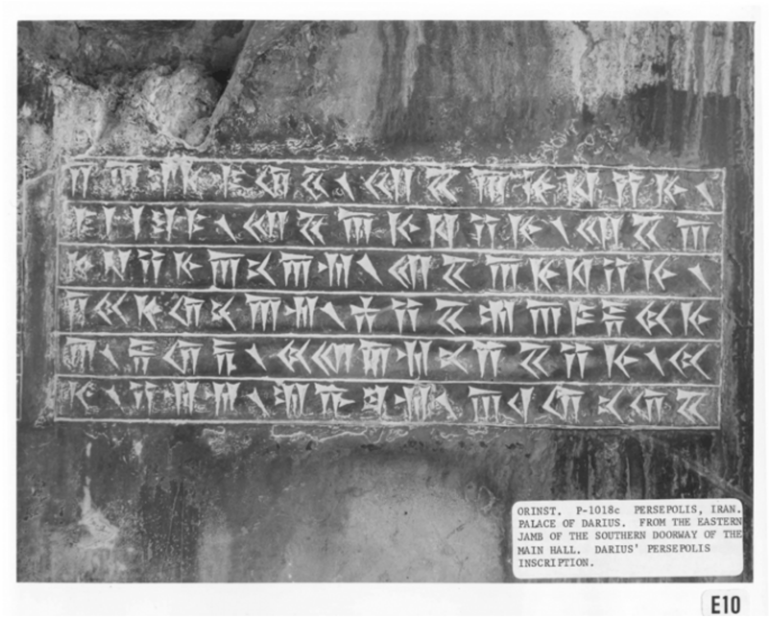
In Arabic calligraphy, the Basmala is a prevalent motif. In Unicode, the Basmala is encoded as one symbol at code point:

```
U+FDFD ARABIC LIGATURE BISMILLAH AR-RAHMAN AR-RAHEEM
```

The Basmala is the Islamic phrase “In the name of God, the Most Gracious, the Most Merciful.”

bi-smi llāhi r-raḥmāni r-raḥīmi
In the name of / Allah / The Beneficent / The Merciful

ب	U+0628	ARABIC LETTER BEH
ِ	U+0650	ARABIC KASRA
سْ	U+0633	ARABIC LETTER SEEN
	U+0652	ARABIC SUKUN
م	U+0645	ARABIC LETTER MEM
ِ	U+0650	ARABIC KASRA
	U+0020	SPACE
ٱ	U+0671	ARABIC LETTER ALEF WASLA
ل	U+0644	ARABIC LETTER LAM
ل	U+0644	ARABIC LETTER LAM
َ	U+064e	ARABIC FATHA
ّ	U+0651	ARABIC SHADDA
ٰ	U+0670	ARABIC LETTER SUPERSCRIPT ALEF
ه	U+0647	ARABIC LETTER HEH
ِ	U+0650	ARABIC KASRA
	U+0020	SPACE
ٱ	U+0671	ARABIC LETTER ALEF WASLA
ل	U+0644	ARABIC LETTER LAM
ر	U+0631	ARABIC LETTER REH
َ	U+064e	ARABIC FATHA
ّ	U+0651	ARABIC SHADDA
حْ	U+062d	ARABIC LETTER HAH
	U+0652	ARABIC SUKUN
م	U+0645	ARABIC LETTER MEEM
َ	U+064e	ARABIC FATHA
ٰ	U+0670	ARABIC LETTER SUPERSCRIPT ALEF
ن	U+0646	ARABIC LETTER NOON
ِ	U+0650	ARABIC KASRA
	U+0020	SPACE
ٱ	U+0671	ARABIC LETTER ALEF WASLA
ل	U+0644	ARABIC LETTER LAM
ر	U+0631	ARABIC LETTER REH
َ	U+064e	ARABIC FATHA
ّ	U+0651	ARABIC SHADDA
ح	U+062d	ARABIC LETTER HAH
ِ	U+0650	ARABIC KASRA
ي	U+064A	ARABIC LETTER YEH
م	U+0645	ARABIC LETTER MEEM
ِ	U+0650	ARABIC KASRA


ORINST. P-1018c PERSEPOLIS, IRAN. PALACE OF DARIUS. FROM THE EASTERN JAMB OF THE SOUTHERN DOORWAY OF THE MAIN HALL. DARIUS' PERSEPOLIS INSCRIPTION.

𐎬	𐎠	𐎼	𐎹	𐎺	𐎢	𐏁	𐏐	𐎧	𐏁	𐎠	𐎹	𐎰	𐎡	𐎹	𐏐
U+103AC	U+103A0	U+103BC	U+103B9	U+103BA	U+103A2	U+103C1	U+103D0	U+103A7	U+103C1	U+103A0	U+103B9	U+103B0	U+103A1	U+103B9	U+103D0
da	a	ra	ya	va	u	sha	\	xa	sha	a	ya	tha	i	ya	\

𐎺	𐏀	𐎼	𐎣	𐏐	𐎧	𐏁	𐎠	𐎹	𐎰	𐎡	𐎹	𐏐	𐎧	𐏁	𐎠
U+103BA	U+103C0	U+103BC	U+103A3	U+103D0	U+103A7	U+103C1	U+103A0	U+103B9	U+103B0	U+103A1	U+103B9	U+103D0	U+103A7	U+103C1	U+103A0
va	za	ra	ka	\	xa	sha	a	ya	tha	i	ya	\	xa	sha	a

𐎹	𐎰	𐎡	𐎹	𐎠	𐎴	𐎠	𐎶	𐏐	𐎧	𐏁	𐎠	𐎹	𐎰	𐎡	𐎹	𐏐
U+103B9	U+103B0	U+103A1	U+103B9	U+103A0	U+103B4	U+103A0	U+103B6	U+103D0	U+103A7	U+103C1	U+103A0	U+103B9	U+103B0	U+103A1	U+103B9	U+103D0
ya	tha	i	ya	a	na	a	ma	\	xa	sha	a	ya	tha	i	ya	\

𐎭	𐏃	𐎹	𐎢	𐎴	𐎠	𐎶	𐏐	𐎻	𐎡	𐏁	𐎫	𐎠	𐎿	𐎱	𐏃	𐎹
U+103AD	U+103C3	U+103B9	U+103A2	U+103B4	U+103A0	U+103B6	U+103D0	U+103BB	U+103A1	U+103C1	U+103AB	U+103A0	U+103BF	U+103B1	U+103C3	U+103B9
da	ha	ya	u	na	a	ma	\	vi	i	sha	ta	a	sa	pa	ha	ya

Darius / king / great / king / king of kings/ king / of the provinces / of Vistaspes/ son / the Achaemenid / who / this palace / made

Darius the Great King, King of Kings, King of countries, son of Hystaspes, an Achaemenian, built this palace.

Happy New Year!

新	年	快	乐
U+65B0	U+5E74	U+5FEB	U+4E50
new year		happiness	

In Mandarin (Pinyin): xīn nián kuài lè
In Cantonese (Jyutping): san^1 nin^4 faai^3 lok^6

Sunday, 22 Jan 2023, year of the rabbit
Saturday, 10 Feb 2024, year of the dragon

CKJV Unification

Because of the enormous number of Asian glyphs, a saving of space can be achieved by unifying glyphs as single ideographs.

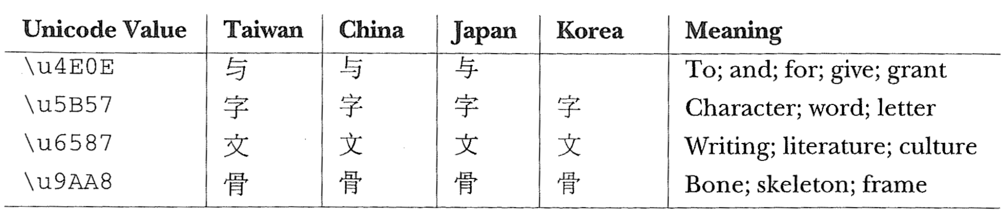


Table 6-10. Unified Characters Requiring Different Glyphs

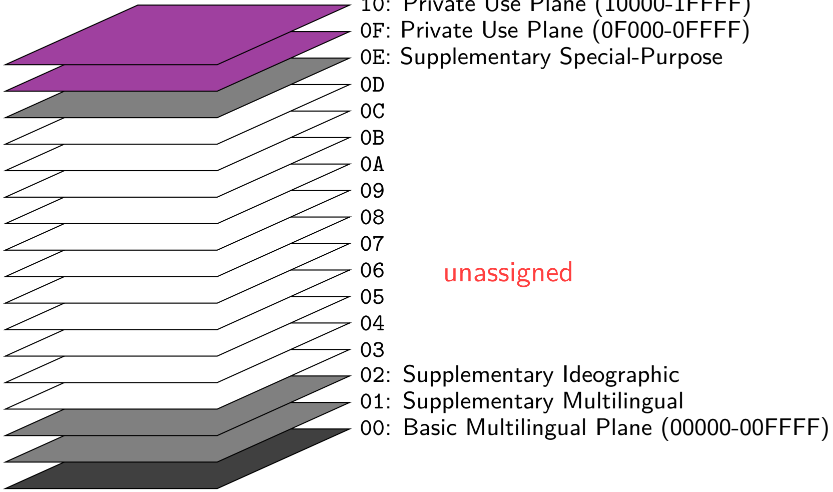
Unicode Value	Taiwan	China	Japan	Korea	Meaning
\u4E0E	与	与	与		To; and; for; give; grant
\u5B57	字	字	字	字	Character; word; letter
\u6587	文	文	文	文	Writing; literature; culture
\u9AA8	骨	骨	骨	骨	Bone; skeleton; frame

16 bits?

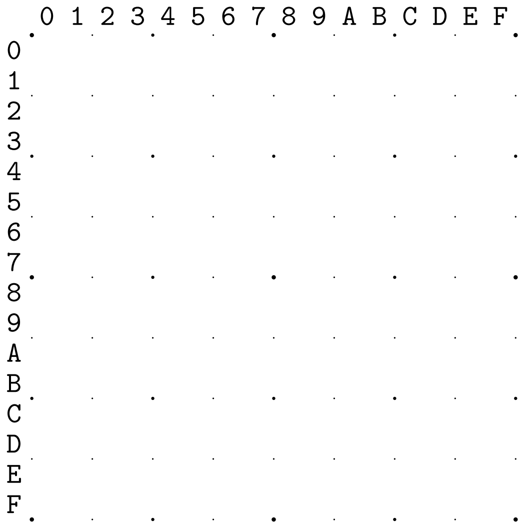
Unicode originally planned a 16 bit encoding. So the `char` data type in Java is 16 bits. But soon there were too many characters. Now Unicode encodes characters into a 21 bit space.

For all practical purposes it is possible work in what is called the 16 bit “base multi-lingual plane.” However, in Java you cannot assume that one char is one Unicode character. See UTF-16.

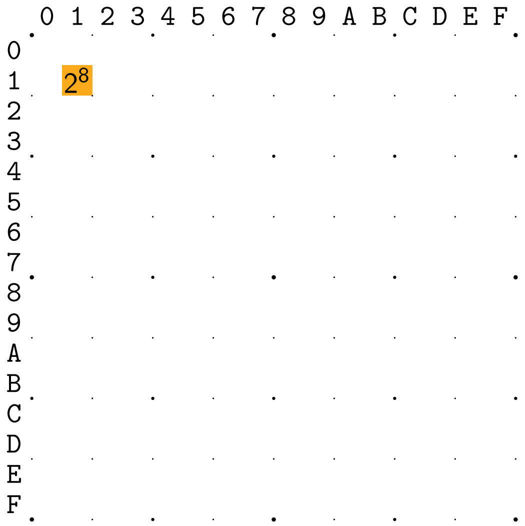
17 Unicode Planes


10: Private Use Plane (10000-1FFFF)
0F: Private Use Plane (0F000-0FFFF)
0E: Supplementary Special-Purpose
0D
0C
0B
0A
09
08
07
06
unassigned
05
04
03
02: Supplementary Ideographic
01: Supplementary Multilingual
00: Basic Multilingual Plane (00000-00FFFF)

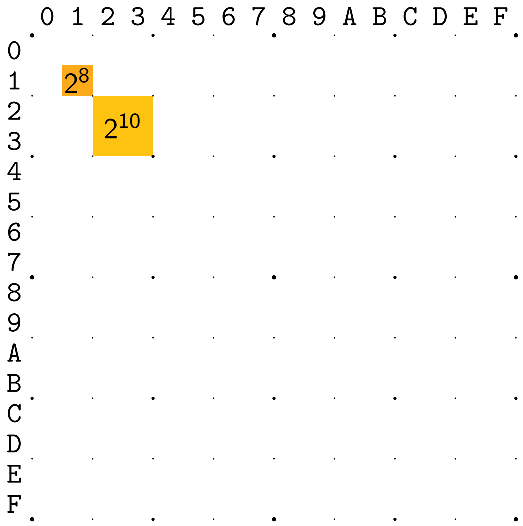
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F

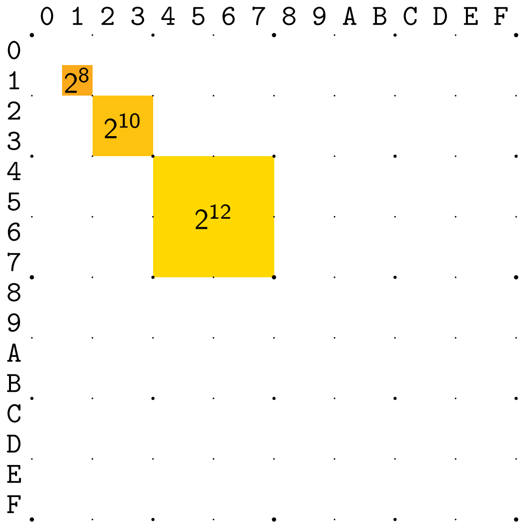
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
2^8

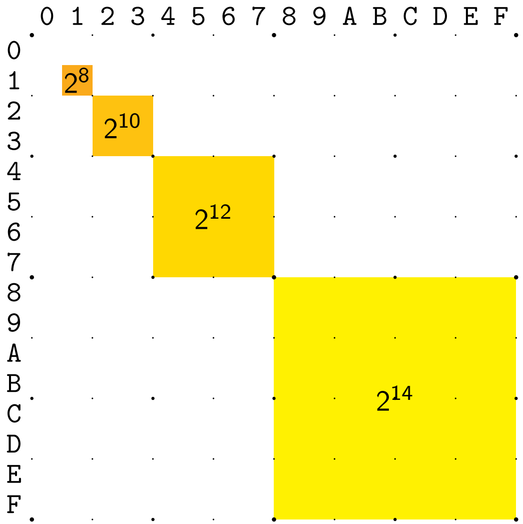
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
2^8
2^{10}

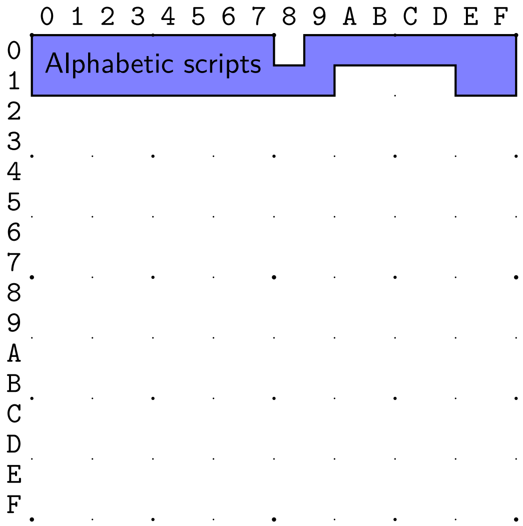
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
2^8
2^{10}
2^{12}

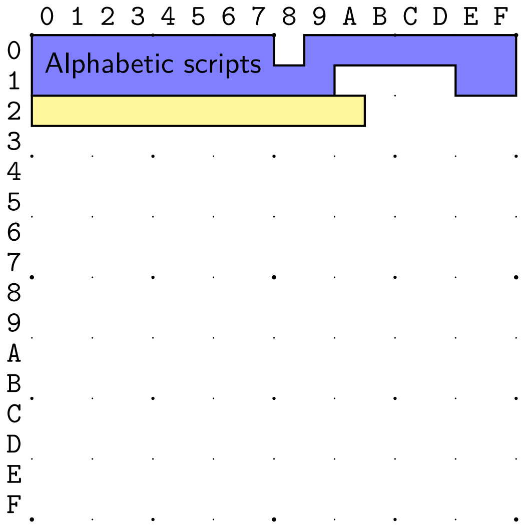
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
2^8
2^{10}
2^{12}
2^{14}

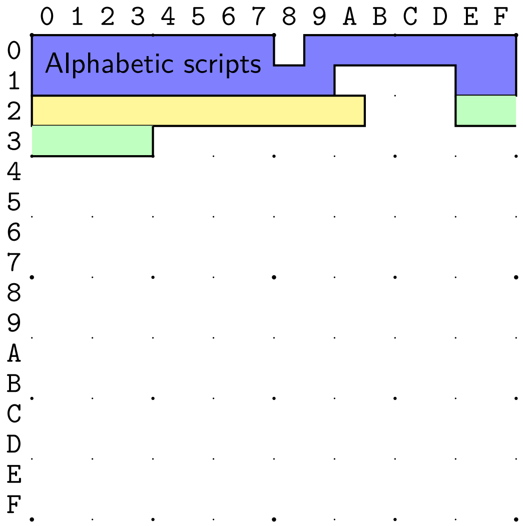
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts

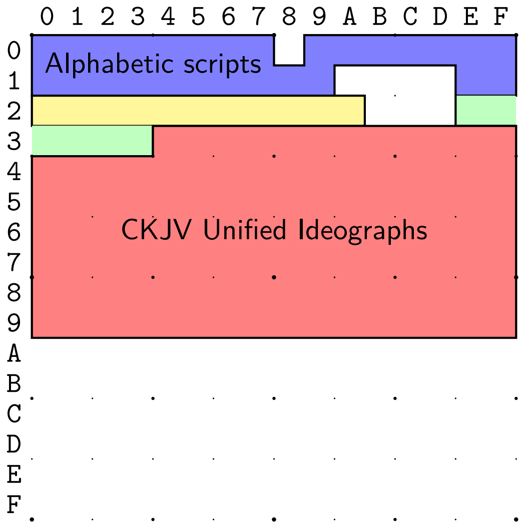
Basic Multilingual Plane (U+0000 – U+FFFF)



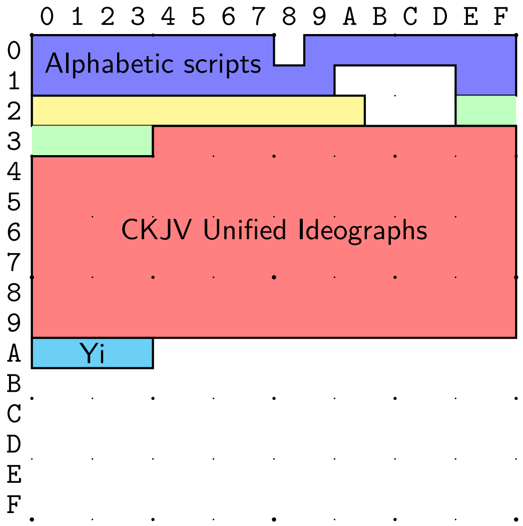
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts

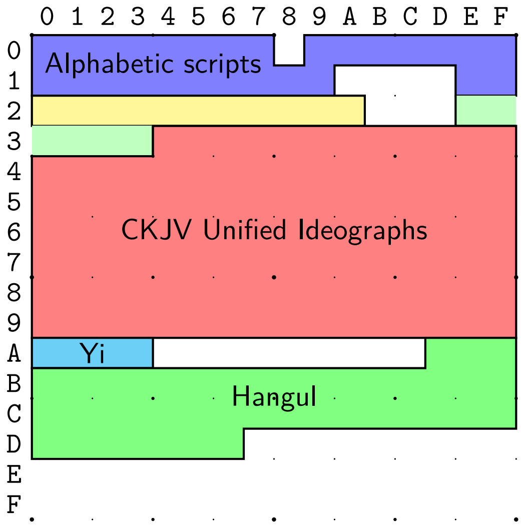
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts
CKJV Unified Ideographs

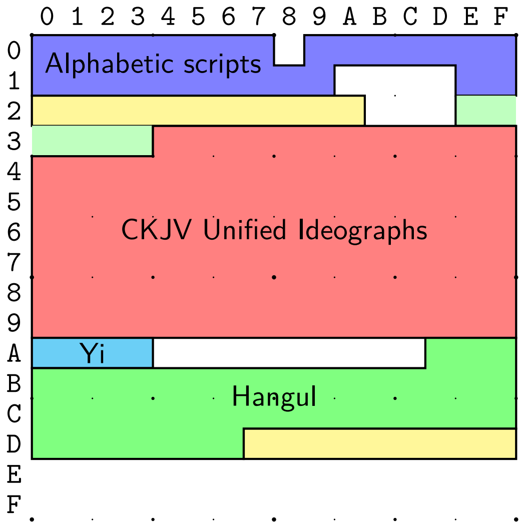
Basic Multilingual Plane (`U+0000` – `U+FFFF`)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts
CKJV Unified Ideographs
Yi

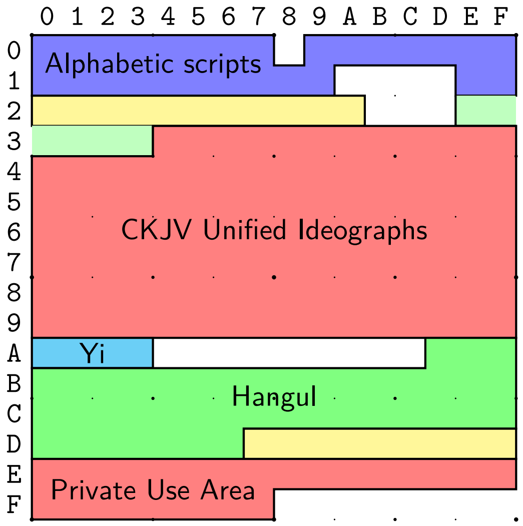
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts
CKJV Unified Ideographs
Yi
Hangul

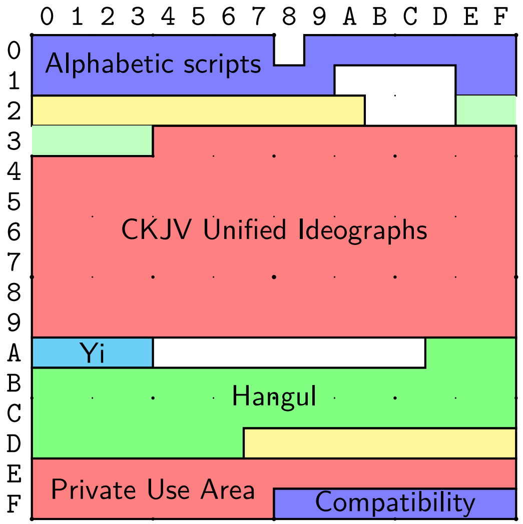
Basic Multilingual Plane (U+0000 – U+FFFF)


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
Alphabetic scripts
CKJV Unified Ideographs
Yi
Hangul

Basic Multilingual Plane (U+0000 – U+FFFF)



Basic Multilingual Plane (`U+0000` – `U+FFFF`)



Scripts Area in Unicode

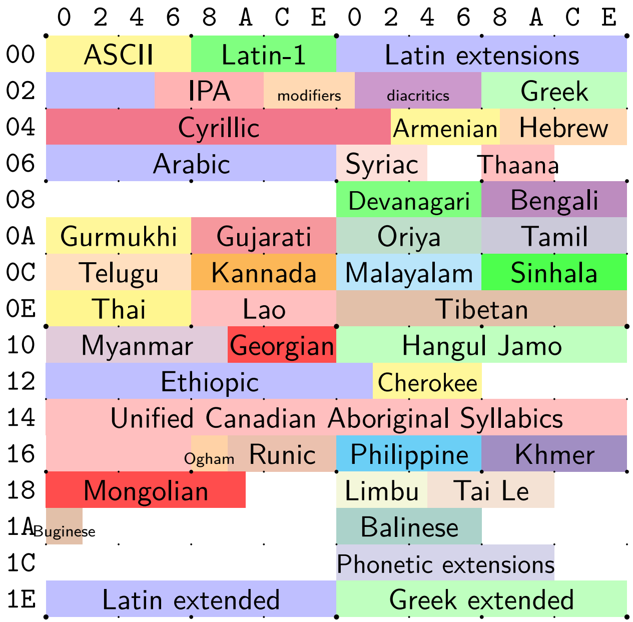
Scripts supported `http://unicode.org/standard/supported.html`

Road map: `http://www.unicode.org/roadmaps/bmp/`

Greek and Coptic U0370 U03FF
`https://en.wikipedia.org/wiki/Greek_alphabet`; Greek Extended U1F00 U1FFF

Buginese/Lontara U1A00 U1A1F
`https://en.wikipedia.org/wiki/Lontara_alphabet` Ogham U1680 U169F
`https://en.wikipedia.org/wiki/Ogham`

space modifiers	U02B0 U02FF
diacritical marks	U0300 U036F
Tai Le/Dehong Dai	U1950 U197F


0 2 4 6 8 A C E 0 2 4 6 8 A C E
00 ASCII Latin-1 Latin extensions
02 IPA modifiers diacritics Greek
04 Cyrillic Armenian Hebrew
06 Arabic Syriac Thaana
08 Devanagari Bengali
0A Gurmukhi Gujarati Oriya Tamil
0C Telugu Kannada Malayalam Sinhala
0E Thai Lao Tibetan
10 Myanmar Georgian Hangul Jamo
12 Ethiopic Cherokee
14 Unified Canadian Aboriginal Syllabics
16 Ogham Runic Philippine Khmer
18 Mongolian Limbu Tai Le
1A Buginese Balinese
1C Phonetic extensions
1E Latin extended Greek extended

String versus Text

Because much of computing is processing written text. We need the best possible primitive data structures.

We can accept the hard work of the Unicode Consortium to provide an inventory of symbols. But even the smallest of steps with natural language are fraught. What is a word? How are words organized?

The Unicode Standard and Java provide support for text processing. It is important to understand this as different than string processing which not aware of the cultural differences that impact written text processing.

Collating Sequence

It is natural to use the code point of a character as the order of the characters. That is, the code point (a number) is taken as the position in the collating sequence of the characters. This extends naturally (via lexicographic ordering) to strings of characters.

Dictionary Order

However, lexicographic ordering on strings of character code points, does not meet cultural expectations. String of characters are not quite the same a words of natural language text.

péche
pêchi
péché
pécher
pêcher

In French péché should follow pêche in a sorted list which it would not by rules of the English language.

Searching and Sorting

Quest For The Symbol to Represent Powerset

In text there are many italic, slanted, and cursive glyphs for letters. These numerous variations are not really *character* distinctions, and so are not usually given separate Unicode code points.

Unicode has nearly 40 code points for characters which resemble the Latin letter ‘p’.

Even uppercase (capital) ‘P’ and lowercase (small) ‘p’ are similar.

One ’P’ like character is used in mathematics for the power set.

HTML5 Entity	char	Unicode	
`&wp;` or `&weierp;` [1]	$\wp$	U+2118	SCRIPT CAPITAL P [2]
`&#x1D443;` [3]	P	U+1D443	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;`	$\mathfrak{P}$ [4]	U+1D513	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;`	$\mathcal{P}$ [5]	U+1D4AB	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;`	$\mathscr{p}$ [6]	U+1D4C5	MATHEMATICAL SCRIPT SMALL P

[1] Certain, simple elliptic functions are named after the mathematician Karl Weierstrass and denoted $\wp$. This notation dates back at least to the first edition of *A Course of Modern Analysis* by E. T. Whittaker in 1902.

HTML5 Entity	char	Unicode	
`&wp;` or `&weierp;` [1]	℘	U+2118	SCRIPT CAPITAL P [2]
`&#x1D443;` [3]	𝑃	U+1D443	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;`	𝔓 [4]	U+1D513	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;`	𝒫 [5]	U+1D4AB	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;`	𝓅 [6]	U+1D4C5	MATHEMATICAL SCRIPT SMALL P

[2] The official Unicode name is a mistake. The Unicode documentation says that U+2118 should have been named "CALLIGRAPHIC SMALL P or perhaps even WEIERSTRASS ELLIPTIC FUNCTION SYMBOL"

HTML5 Entity	char	Unicode	
`&wp;` or `&weierp;` [1]	℘	U+2118	SCRIPT CAPITAL P [2]
`&#x1D443;` [3]	𝑃	U+1D443	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;`	𝔓 [4]	U+1D513	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;`	𝒫 [5]	U+1D4AB	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;`	𝓅 [6]	U+1D4C5	MATHEMATICAL SCRIPT SMALL P

[3] HTML5 does not have a named entity for this ‘p‘. Italic or slanted fonts are the primary way mathematicians can visually distinguish words of text from formulas of mathematics. Modern technology eases the use of more symbols in mathematical writing.

HTML5 Entity	char	Unicode	
`&wp;` ↗ or `&weierp;` ↗ ❶	$\wp$	U+2118 ↗	SCRIPT CAPITAL P ❷
`&#x1D443;` ↗ ❸	P	U+1D443 ↗	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;` ↗	$\mathfrak{P}$ ❹	U+1D513 ↗	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;` ↗	$\mathcal{P}$ ❺	U+1D4AB ↗	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;` ↗	𝓅 ❻	U+1D4C5 ↗	MATHEMATICAL SCRIPT SMALL P

❹ The German "fraktur" script, no longer in use with ordinary text, yields an entirely new set of Latin letter forms from which mathematicians have gleaned symbols.

HTML5 Entity	char	Unicode	
`&wp;` or `&weierp;` ❶	℘	U+2118	SCRIPT CAPITAL P ❷
`&#x1D443;` ❸	𝑃	U+1D443	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;`	𝔓 ❹	U+1D513	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;`	𝒫 ❺	U+1D4AB	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;`	𝓅 ❻	U+1D4C5	MATHEMATICAL SCRIPT SMALL P

❺ There are various mathematical script fonts in LaTeX. The Euler Script symbols were designed by Hermann Zapf and they are included with the standard LaTeX distribution. But they have no lowercase letters.

HTML5 Entity	char	Unicode	
`&wp;` or `&weierp;` (1)	℘	U+2118	SCRIPT CAPITAL P (2)
`&#x1D443;` (3)	𝑃	U+1D443	MATHEMATICAL ITALIC CAPITAL P
`&Pfr;`	𝔓 (4)	U+1D513	MATHEMATICAL FRAKTUR CAPITAL P
`&Pscr;`	𝒫 (5)	U+1D4AB	MATHEMATICAL SCRIPT CAPITAL P
`&pscr;`	𝓅 (6)	U+1D4C5	MATHEMATICAL SCRIPT SMALL P

(6) Many LaTeX mathematical, script fonts do not have lowercase letters. Here is glyph from the Zapf Chancery calligraphic font in LaTeX: *p*. The free Google Noto Sans Math TTF is possibililty here. The glyph shown above comes from FileFormat.Info.

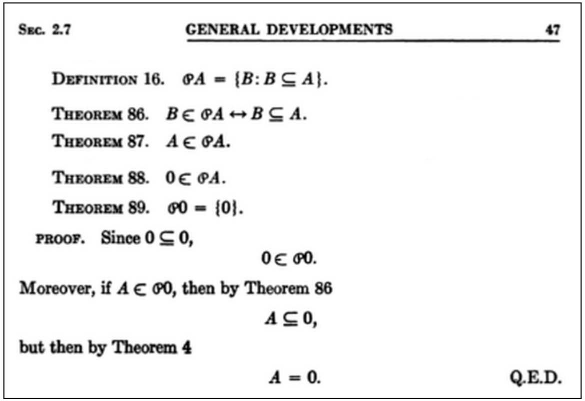
Wikipedia claims: “Starting with Unicode 3.0.1, a separate, capital symbol is available for power set, namely U+1D4AB MATHEMATICAL SCRIPT CAPITAL P (HTML `&#119979;`), which is available as `&Pscr;`.” Starting in version 5.0 Unicode gives POWER SET as an alternate name for the character.

Regardless of the codepoint, no widely available LaTeX font renders the symbol in the way that I would like. Here are some various possibilities for the script P available in LaTeX.

```
$P$ $\mathcal{P}$ $\EuScript{P}$ $\mathscr{P}$ $\mathpzc{P}$
```

$$P(X) \quad \mathcal{P}(X) \quad \mathcal{P}(X) \quad \mathscr{P}(X) \quad \mathcal{P}(X)$$

In writing by hand, I always mimic the calligraphic letter I saw in books like *Axiomatic Set Theory* by Suppes or *The Theory of Parsing, Translation, and Compiling. Volume 1: Parsing* by Aho and Ullman. The same symbol was used by Greibach in the JACM, but for a different purpose.



DEFINITION 16. $\mathcal{P}A = \{B : B \subseteq A\}$.

THEOREM 86. $B \in \mathcal{P}A \leftrightarrow B \subseteq A$.

THEOREM 87. $A \in \mathcal{P}A$.

THEOREM 88. $0 \in \mathcal{P}A$.

THEOREM 89. $\mathcal{P}0 = \{0\}$.

PROOF. Since $0 \subseteq 0$,

$$0 \in \mathcal{P}0.$$

Moreover, if $A \in \mathcal{P}0$, then by Theorem 86

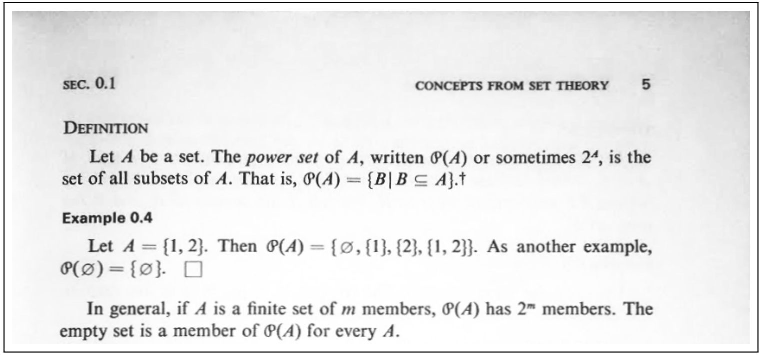
$$A \subseteq 0,$$

but then by Theorem 4

$$A = 0.$$

Q.E.D.

Patrick Suppes, *Axiomatic Set Theory*, D. van Nostrand, 1960, page 47.



SEC. 0.1 CONCEPTS FROM SET THEORY 5

DEFINITION

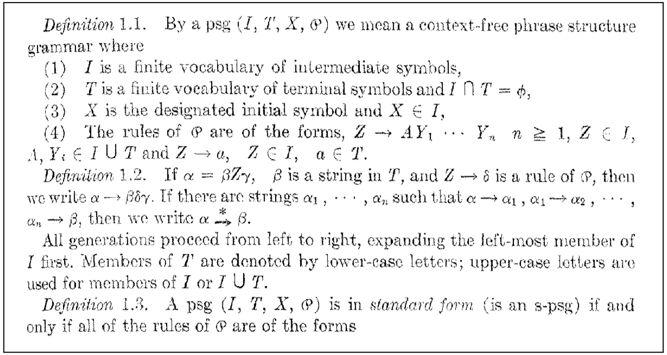
Let A be a set. The *power set* of A, written $\mathcal{P}(A)$ or sometimes 2^A, is the set of all subsets of A. That is, $\mathcal{P}(A) = \{B \mid B \subseteq A\}$.†

Example 0.4

Let $A = \{1, 2\}$. Then $\mathcal{P}(A) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\}$. As another example, $\mathcal{P}(\varnothing) = \{\varnothing\}$. $\square$

In general, if A is a finite set of m members, $\mathcal{P}(A)$ has 2^m members. The empty set is a member of $\mathcal{P}(A)$ for every A.

Alfred Vaino Aho and Jeffrey David Ullman, *The Theory of Parsing, Translation, and Compiling*. Volume 1: *Parsing*, Prentice-Hall, 1972, page 5.



Definition 1.1. By a psg $(I, T, X, \mathcal{P})$ we mean a context-free phrase structure grammar where

(1) I is a finite vocabulary of intermediate symbols,

(2) T is a finite vocabulary of terminal symbols and $I \cap T = \phi$,

(3) X is the designated initial symbol and $X \in I$,

(4) The rules of $\mathcal{P}$ are of the forms, $Z \to AY_1 \cdots Y_n \quad n \geqq 1$, $Z \in I$, $A, Y_i \in I \cup T$ and $Z \to a, \quad Z \in I, \quad a \in T$.

Definition 1.2. If $\alpha = \beta Z \gamma, \quad \beta$ is a string in T, and $Z \to \delta$ is a rule of $\mathcal{P}$, then we write $\alpha \to \beta\delta\gamma$. If there are strings $\alpha_1, \cdots, \alpha_n$ such that $\alpha \to \alpha_1, \alpha_1 \to \alpha_2, \cdots, \alpha_n \to \beta$, then we write $\alpha \overset{*}{\to} \beta$.

All generations proceed from left to right, expanding the left-most member of I first. Members of T are denoted by lower-case letters; upper-case letters are used for members of I or $I \cup T$.

Definition 1.3. A psg $(I, T, X, \mathcal{P})$ is in *standard form* (is an s-psg) if and only if all of the rules of $\mathcal{P}$ are of the forms

Sheila A. Greibach, "A New Normal-Form Theorem for Context-Free Phrase Structure Grammars," *J. ACM*, volume 12, number 1, January 1964, pages 42–52.

In LaTeX I use a PNG file scanned from the book by Aho and Ullman.

$$x \in \wp(A)$$

$$x \in \wp(\emptyset)$$

This symbol is easily distinguished from other uses of ‘P’ in mathematics.

$P(A)$	probability of event A
$P \Rightarrow Q$	proposition P implies Q
$\{P\}\ S\ \{Q\}$	P is the precondition

Math Notation

HTML	char	Unicode	
&#x2102	ℂ	U+2102	DOUBLE-STRUCK CAPITAL C
			= the set of complex numbers
&#x2107	ℇ	U+2107	EULER CONSTANT
&#x2135	ℵ	U+2135	ALEF SYMBOL
			= first transfinite cardinal (countable)
&#x2148	ⅈ	U+2148	DOUBLE-STRUCK SMALL I
			• sometimes used for the imaginary unit
&#x1d456	𝑖	U+1D456	MATHEMATICAL ITALIC SMALL I

The very odd-looking "e" associated with Euler's constant in Unicode is never used in mathematics and easily confused with Euler's number.
Euler's number $e = 2.718$ (not to be confused with Euler's constant $\gamma = 0.577$) and the "i" in imaginary numbers do not generally receive distinguishing font treatment in mathematical typography.

Logic Symbols

Bocheński	char	LaTeX	Unicode	
A	$\vee$	`\vee`	U+2228	LOGICAL OR
K	$\wedge$	`\wedge`	U+2227	LOGICAL AND
	$\oplus$	`\oplus`	U+2295	CIRCLED PLUS
J	$\veebar$	`\veebar`	U+22BB	XOR
	$\nLeftrightarrow$	`\nLeftrightarrow`	U+21F9	LEFT RIGHT ARROW WITH VERTICAL STROKE
D	$\barwedge$	`\barwedge`[1]	U+22BC	NAND
	$\uparrow$	`\uparrow`	U+2191	UPWARDS ARROW
X	$\bar\vee$	`\bar\vee`	U+22BD	NOR
	$\downarrow$	`\downarrow`	U+2193	DOWNWARDS ARROW

The command $\barwedge$ and $\veebar$ come from the LaTeX `amsymb` package. The command `\land`, `\lor` and `\lnot` are synonyms for: `\wedge`, `\vee` and `\neg`.

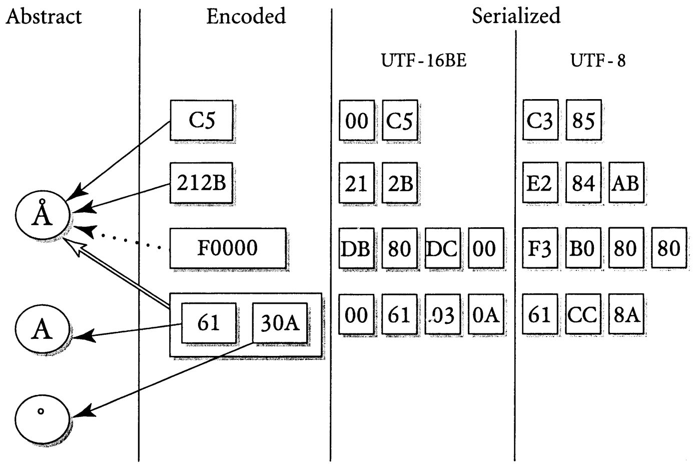
Encoding Schemes

The enormous number of symbols in the Unicode inventory cause concern. It is desirable to avoid the awkwardness of computing with 21-bits worth of symbols, when that majority of processing using only small subsets of the vast Unicode inventory.

Information scientists have created strategies to manage this big inventory and this techniques are in widespread use.

Encoding Schemes

Figure 2-6. Character Encoding Schemes



UTF-8, UTF-16

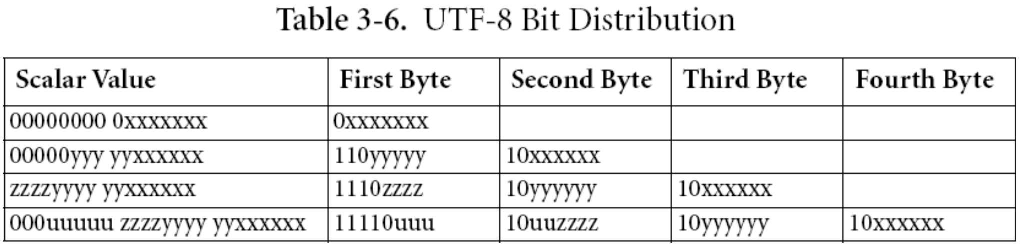


Table 3-6. UTF-8 Bit Distribution

Scalar Value	First Byte	Second Byte	Third Byte	Fourth Byte
00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

This makes US-ASCII automatically UTF-8 – all seven bit characters are encoded in 8 bits. (11 bits in 16 bits; 16 bits in 24 bits; all 21 bits in 32 bits.)

UTF-16, UTF-32

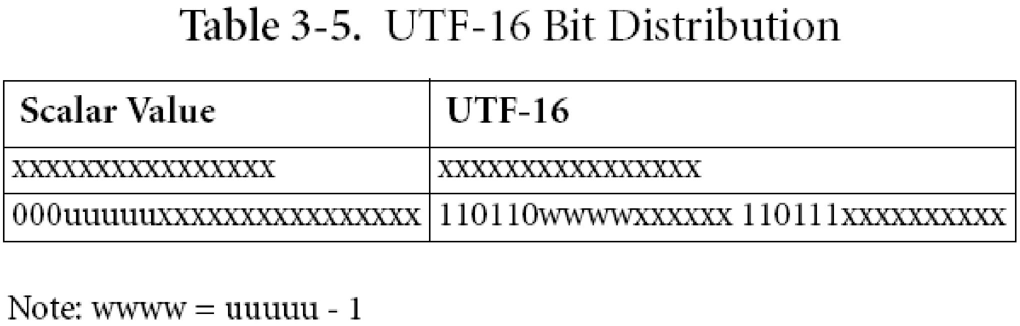


Table 3-5. UTF-16 Bit Distribution

Scalar Value	UTF-16
xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
000uuuuuxxxxxxxxxxxxxxxx	110110wwwwxxxxxx 110111xxxxxxxxxx

Note: wwww = uuuuu - 1

Java

Convert an array of int (code points) to UTF-16.

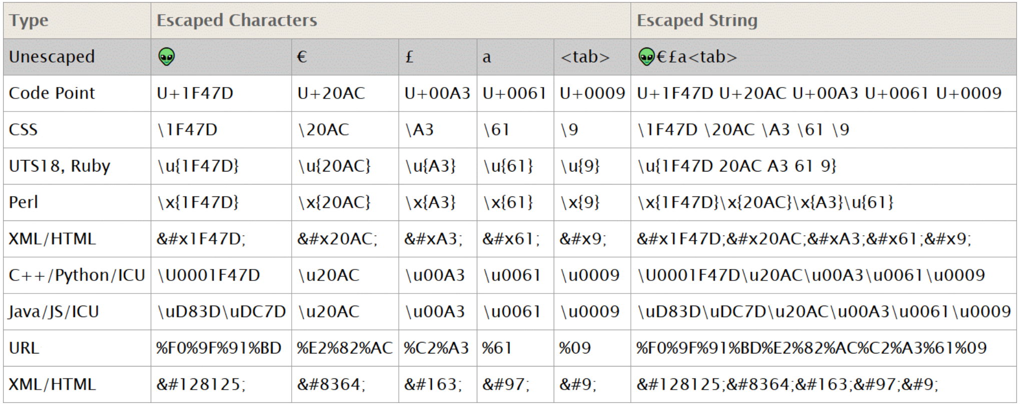
```
String UTF16 = new String (ints, 0, ints.length);
```

Python

Convert a list of bytes to an internal representation of Unicode which varies in Python (Latin-1, UTF-16, UTF-32) on a string-by-string basis.

```
>>> str(bytes([0x24]),'utf8') == '$'
>>> str(bytes([0xE0,0xA4,0xB9]),'utf8') == '\u0939'
>>> str(bytes([0xED,0x95,0x9C]),'utf8') == '\uD55C'
>>> str(bytes([0xF0,0x90,0x8D,0x88]),'utf8') == '\U00010348'

>>> str(bytes([0x00,0x24]),'utf-16-be') == '$'
>>> str(bytes([0x24,0x00]),'utf-16-le') == '$'
>>> str(bytes([0x20,0xAC]),'utf-16-be') == '\u20ac'
>>> str(bytes([0xAC,0x20]),'utf-16-le') == '\u20ac'
>>> str(bytes([0xD8,0x01,0xDC,0x37]),'utf-16-be')=='\u00010437'
>>> str(bytes([0x01,0xD8,0x37,0xDC]),'utf-16-le')=='\u00010437'
>>> str(bytes([0xD8,0x52,0xDF,0x62]),'utf-16-be')=='\u00024B62'
>>> str(bytes([0x52,0xD8,0x62,0xDF]),'utf-16-le')=='\u00024B62'
```



Type	Escaped Characters					Escaped String
Unescaped	👽	€	£	a	<tab>	👽€£a<tab>
Code Point	U+1F47D	U+20AC	U+00A3	U+0061	U+0009	U+1F47D U+20AC U+00A3 U+0061 U+0009
CSS	\1F47D	\20AC	\A3	\61	\9	\1F47D \20AC \A3 \61 \9
UTS18, Ruby	\u{1F47D}	\u{20AC}	\u{A3}	\u{61}	\u{9}	\u{1F47D 20AC A3 61 9}
Perl	\x{1F47D}	\x{20AC}	\x{A3}	\x{61}	\x{9}	\x{1F47D}\x{20AC}\x{A3}\u{61}
XML/HTML	&#x1F47D;	&#x20AC;	&#xA3;	&#x61;	&#x9;	&#x1F47D;&#x20AC;&#xA3;&#x61;&#x9;
C++/Python/ICU	\U0001F47D	\u20AC	\u00A3	\u0061	\u0009	\U0001F47D\u20AC\u00A3\u0061\u0009
Java/JS/ICU	\uD83D\uDC7D	\u20AC	\u00A3	\u0061	\u0009	\uD83D\uDC7D\u20AC\u00A3\u0061\u0009
URL	%F0%9F%91%BD	%E2%82%AC	%C2%A3	%61	%09	%F0%9F%91%BD%E2%82%AC%C2%A3%61%09
XML/HTML	&#128125;	&#8364;	&#163;	&#97;	&#9;	&#128125;&#8364;&#163;&#97;&#9;

	alien	euro	pound	a	<tab>
C++/Python	\U0001F47D	\u20AC	\u00A3	\u0061	\u0009
Java	\uD83D\uDc7D [utf-16]	\u20AC	\u00A3	\u0061	\u009
XML/HTML (hex)	&#x1F47D;	&#x20AC;	&#xA3;	&#x61;	&#x9;
(decimal)	&#128125;	&#8364;	&#163;	&#97;	&#9;
(name)		&euro;	&pound;		&Tab;

UTF-8 Security

Latin1 is not the same as UTF-8. All bytes are legal Latin1 if you include control characters. Not all sequences of bytes are valid UTF-8

- invalid bytes 41 ['A'] FE 5A ['Z'] (the bytes in the range F5-FF cannot occur)
- an unexpected continuation byte 41 ['A'] 80 BF 5A ['Z'] (the byes in the range 80-BF are continuation bytes)
- a string ending too soon (lonely start) 41 ['A'] C0 E0 5A ['Z']
- on overlong encoding 41 ['A'] F0 82 82 AC 5A ['Z']
- a sequence that decodes to an invalid code point

Python

```
# valid UTF-8 sequence
str(bytes([0x41,0xE0,0xA4,0xB9,0x5A]),'utf8') == 'A\u0939Z'
# In Latin-1 0xFE "latin small letter thorn"
# byte must never appears in utf8 sequence
str(bytes([0x41,0xFE,0x5A]),'utf8')
str(bytes([0x41,0xFF,0x5A]),'utf8')
# Normal '/'
str(bytes([0x41,0x2F,0x5A]),'utf8') == 'A/Z'

# overlong
str(bytes([0x41,0xC0,0xAF,0x5A]),'utf8') != 'A/Z'
str(bytes([0x41,0xE0,0x80,0xAF,0x5A]),'utf8') != 'A/Z'
str(bytes([0x41,0xF0,0x80,0x80,0xAF,0x5A]),'utf8') != 'A/Z'
str(bytes([0x41,0xF8,0x80,0x80,0x80,0xAF,0x5A]),'utf8') != 'A/Z'
str(bytes([0x41,0xFC,0x80,0x80,0x80,0x80,0xAF,0x5A]),'utf8') != 'A/Z'

# UTF-16 surrogate not a valid code point
str(bytes([0xED,0xA0,0x80]),'utf8') != '\uD800'
```

UTF-16 Security

Since the ranges for the high surrogates (0xD800–0xDBFF), low surrogates (0XDC00–0XDFFF), and valid BMP characters (0X0000–0xDFFF, 0xE0000–0XFFFF) are disjoint, it is not possible for a surrogate to match a BMP character, or for two adjacent code units to look like a legal surrogate pair. This simplifies searches a great deal. It also means that UTF-16 is self-synchronizing on 16-bit words: whether a code unit starts a character can be determined without examining earlier code units (i.e., the type of code unit can be determined by the ranges of values in which it falls).

As someone who works a lot at the byte <–> Unicode boundary the idea of having strings with an internal UTF-8 encoding is very interesting. Having worked with Rust for a while now I am getting more and more convinced that the approach is a good idea. While it forces you to give up on the idea of being able to address characters individually, that is actually not a huge loss. For a start Unicode would pretty much require you to normalize your strings anyways before you do text processing due to the many ways in which you can format the strings. For instance umlauts come in combined characters but they can also be manually created by placing the regular letter followed by the combining diaeresis character.

Blog by Armin Ronacher.

Byte Order Mark

Text files do not usually have a magic number or other indication of the intended character encoding. However, Unicode does define a BOM.

UTF-8	EF BB BF
UTF-16 (BE)	FE FF
UTF-16 (LE)	FF FE
UTF-32 (BE)	00 00 FE FF
UTF-32 (LE)	FF FE 00 00

So, this is useless knowledge. The point is that *you can't tell the character encoding of a text file by looking at the bytes*. See other file formats like HTML5 and MIME.

Wikipedia reports that UTF8 overtook ASCII as the most common encoding in 2006 and is by far the most common encoding today.

Some argue that UTF-8 is better than UTF-16 that been used in Java, Python, and C++.

- Big-endian versus Little-endian: UTF-16BE, UTF-16LE
- UTF-8 and UTF-32 sort the same lexicographically, UTF-16 does not
- UTF-16 is not a fixed width encoding

See UTF8 Everywhere

Summary

Character encoding standards.

- 7 bits. US-ASCII encoding
- 8 bits. Latin-1, or Latin-0 encoding. UTF-8 multi-byte
- 16 bits. Java `char`, UTF-16 multi-byte
- 18 bits. Required for Unicode code points
- 21 bits. Unicode defines a code-space of 1,114,112 code points in the range 0_{16} to $10FFFF_{16}$.
- 32 bits. UTF-32 or UCS-4 – only fixed-width Unicode encoding

Questions

1. Describe something Java does to make working with different character encodings easier.
2. How might a Java program execute differently on two different computers?