Written Corpora

Master I — TAL
September 2024 - November 2024

Encodings

Bruno Guillaume
Sémagramme Team
LORIA
Bruno.Guillaume@loria.fr

LES TONTONS FLINGUEURS (1963)

Sc@nario d'Albert Simonin
Dialogues de Michel Audiard

LOGO DE LA SOCI@T@ GAUMONT

USINE DE MONTAUBAN - EXT@RIEUR NUIT

Le ciel gris du jour qui va bient@t se coucher, au-dessus de la cour de l'usine de mat@riel de Travaux Publics de Fernand Naudin @ Montauban. Une pelleteuse @ chenilles sort lentement d'un hangar, avec le godet en position haute. Derri@re la pelleteuse apparai@t Monsieur Fernand Naudin, le directeur de l'usine, en grande discussion avec un ouvrier. Il porte un costume gris tr@s @l@gant avec chemise et cravate, et un manteau noir, lui aussi tr@s @l@gant.

MONSIEUR FERNAND
C'est quand m@me pas la premi@re fois, non ?

PREMIER OUVRIER
Je dis pas que c'est la premi@re fois que vous montez @ Paris, Monsieur Fernand, je dis que @a tombe mal. Si le vent est frisquet, vous avez une couverture @ l'arri@re et Germaine a mis du th@ dans le thermos.

MONSIEUR FERNAND
Et pourquoi pas de la quinine et un passe-montagne ? On croirait vraiment que je pars au Tibet.

Il s'approche de sa voiture, une Peugeot 404 noire, qu'un autre ouvrier vient d'amener dans la cour. Pendant qu'il enl@ve son manteau pour @tre plus @ l'aise pour conduire, le premier ouvrier se pr@cipite vers la voiture. Le deuxi@me ouvrier sort de la voiture.

https://imsdb.com/scripts/Les-Tontons-Flingueurs.html

Encodings

How to store a natural language text in a file?

A text written in natural language	English
Un texte écrit en langue naturelle	French
Текст, написанный на естественном языке	Russian
用自然語言寫的文本	Chinese
Բնական լեզվով գրված տեքստը	Armenian
இயற்கை மொழியில் எழுதப்பட்ட ஒரு உரை	Tamil
نص مكتوب بلغة طبيعية	Arabic

```
0000000 e7 94 a8 e8 87 aa e7 84 b6 e8 aa 9e e8 a8 80 e5
0000010 af ab e7 9a 84 e6 96 87 e6 9c ac 0a
```

Chinese

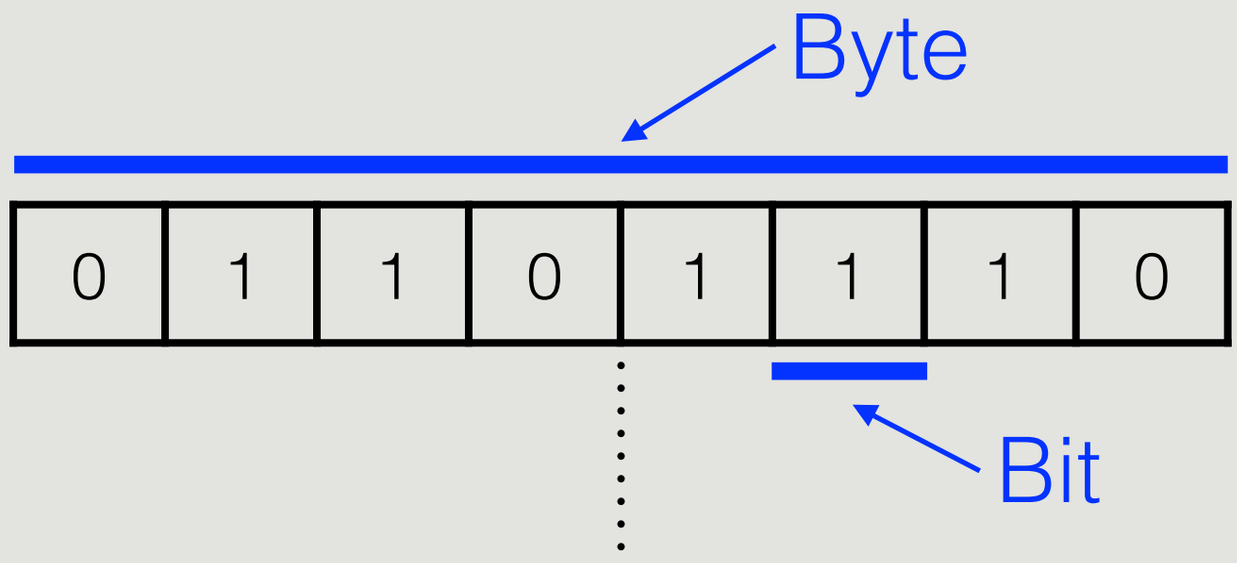
ASCII (1960)

American Standard Code for Information Interchange
128 codes (7 bits), 95 printable characters

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

Binary, decimal & hexadecimal

Binary (2)



Hexadecimal (16)

6 E

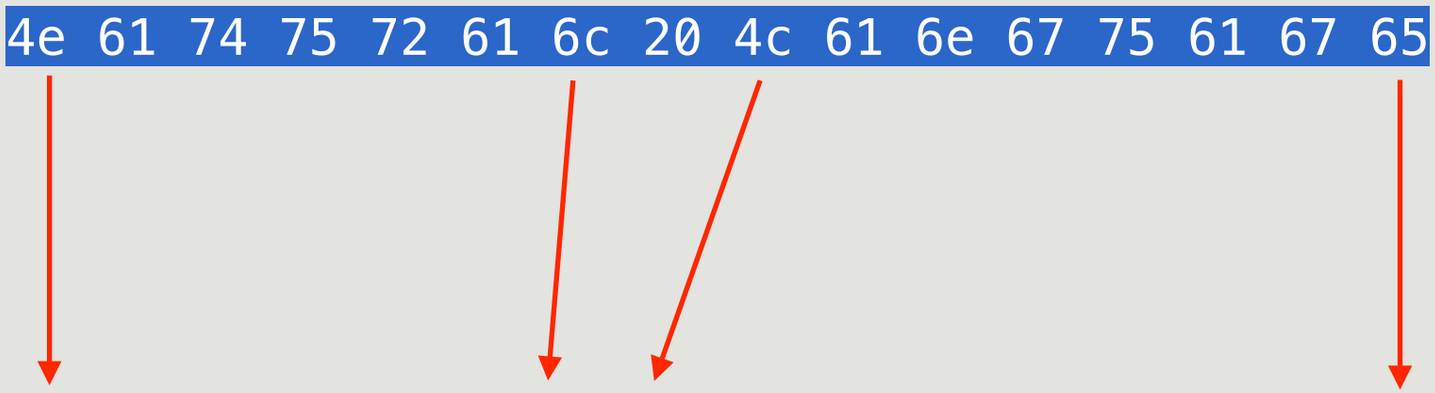
Decimal (10)

$$110 = 6 \times 16 + 14$$

n in ASCII

ASCII

One byte → One Character



Natural Language

Note that the first bit is unused (≤127)

NB: 6e = 6E

How to go beyond English?

Use the 128 remaining codes for missing characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Wikipedia

```
56 6f 69 6c e0 20 6c 27 e9 74 e9
```

Voilà l’été

This is the ISO-8859-1 encoding

Other languages?

In 1987, an international standard appears:
first version of Unicode

Everyone in the world should be able to use their own language on phones and computers.

Unicode provides a unique number for every character, no matter what the platform, program, or language is.

Unicode Website

And the other languages?

Unicode defines a code space of 1,114,112 code points in the range 0 to 10FFFF

Last version:

- Unicode 16.0.0 (September 10, 2024)
  - **154,998** characters
    - **5,185** new characters
    - **13.91**% of available code points
  - covering **168** modern and historic scripts
    - Characters Code Charts
- New in version 16.0.0

Unicode

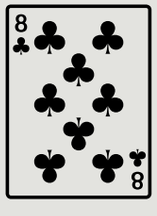
Each character has:

- a unique code point (U+hex)
- a description

n U+006E Latin Small Letter N

 U+1A13 Buginese Letter Va

 U+13037 Egyptian Hieroglyph A046

 U+1F0D8 Playing Card Eight of Clubs

 U+FE18 Presentation Form for Vertical Right White Lenticular Brakcet

Unicode

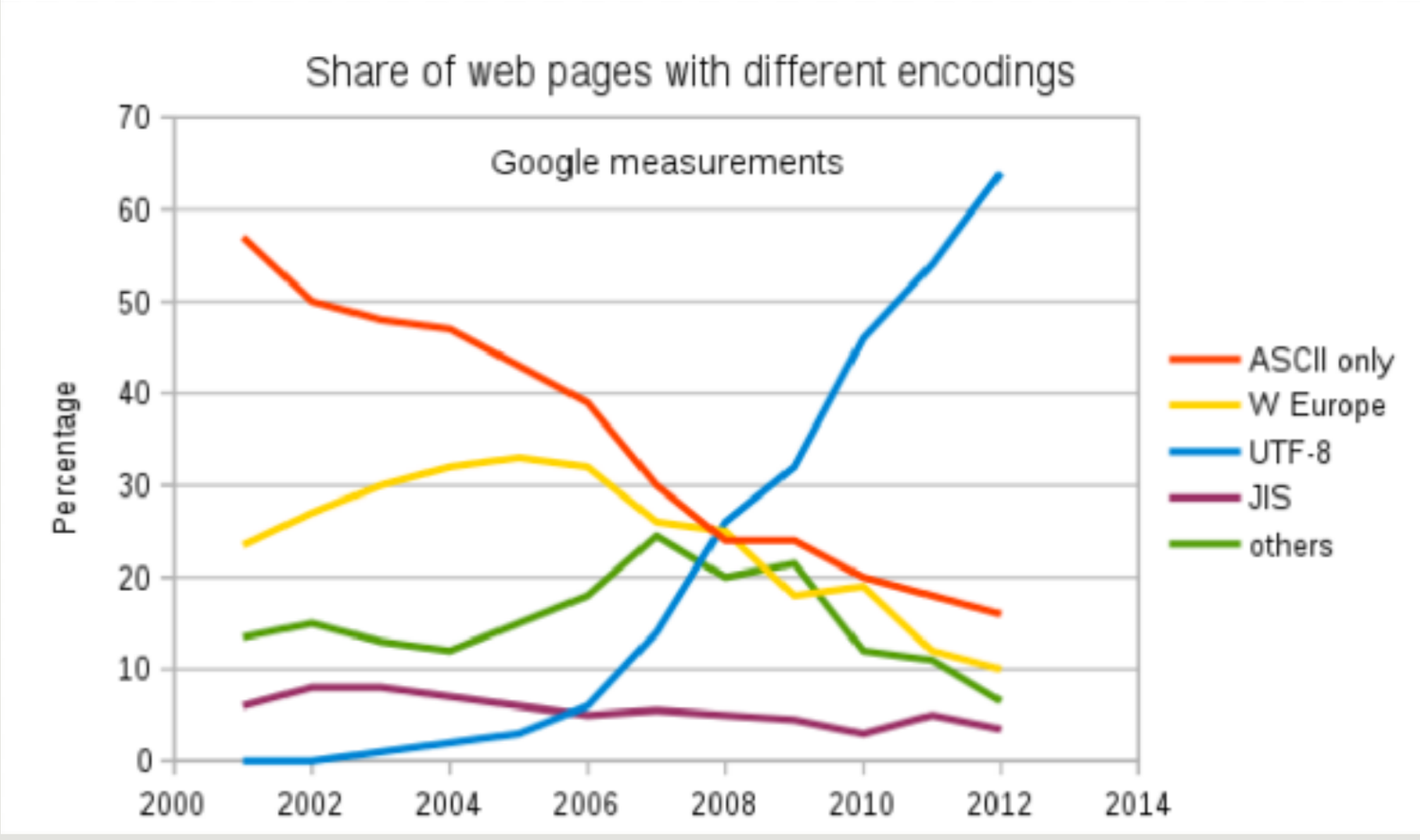
Unicode is a character set
Unicode is **not** an encoding

Unicode specifies also a set of **encodings**
UTF-8, UTF-16, UTF-32

UTF-8 has became a heavily used standard

You **must** use UTF-8 to store data

UTF-8



Wikipedia

https://w3techs.com/technologies/details/en-utf8

UTF-8

UTF-8 is defined to encode code points in one to four bytes

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	`0xxxxxxx`			
2	11	U+0080	U+07FF	`110xxxxx`	`10xxxxxx`		
3	16	U+0800	U+FFFF	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	
4	21	U+10000	U+10FFFF	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

From Unicode to UTF-8:

- U+20AC (€)
- 20AC —> 10 0000 1010 1100 —> 0010 0000 1010 1100
- 1110 0010 1000 0010 1010 1100 —> E2 82 AC

NB: ASCII ⊂ UTF-8

Unix tools

Unix commands:

- `cat file` print the content of the file
- `file file` print info about the file (encoding, newline…) ⚠️ not always reliable
- `diff file1 file2` print difference between file
- `iconv -f ISO-8859-1 -t UTF-8 file1 > file2` encoding conversion
- `iconv -l` list of known encodings
- `dos2unix file` change newlines in the file
- `hexdump -C file` show hexadecimal content and the text (ASCII) in the second columns

Other tools

Most modern tools are UTF-8 based by default

- **Python**: Use Python 3 and avoid Python 2
  (if needed, in Python 2, use `s = u"été"` to declare an UTF-8 string)

- **LaTeX**:

```
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
```

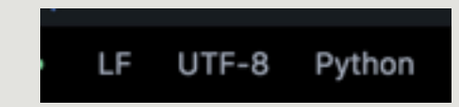
- **HTML-4**:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

- **HTML-5**: The default character encoding is UTF-8.
  The usage of another encoding must be specified `<meta charset="ISO-8859-1">`

- **XML**: `<?xml version="1.0" encoding="UTF-8"?>`

Make sure that your favourite text editor uses UTF-8 by default



Atom
VSCode

Encodings, exercises

https://grew.fr/ee.tgz

(1) Observe the content of the two files `1A.txt` and `1B.txt` (with the `cat` Unix command for example), observe the output of the command `diff 1A.txt 1B.txt`. Explain the difference

(2) Same question with files `2A.txt` and `2B.txt`

(3) Same question with files `3A.txt` and `3B.txt`

(4) Same question with files `4A.txt` and `4B.txt`

(5) Same question with files `5A.txt`, `5B.txt` and `5C.txt`

If you don't have a Unix terminal on your labtop: https://bellard.org/jslinux/

Encodings, exercise 1

They look the same but there are different!

Tabulation

```
 > hexdump -C 1A.txt
0000000 41 09 42 0a
```

Spaces

```
 > hexdump -C 1B.txt
0000000 41 20 20 20 20 20 20 20 42 0a
```

Encodings, exercise 2

They look the same but there are different!

LF: Newline in Unix

```
 > hexdump -C 2A.txt
0000000 41 0a 42 0a
```

dos2unix

CRLF: Newline in Windows/DOS

```
 > hexdump -C 2B.txt
0000000 41 0d 0a 42 0d 0a
```

9	9	[HORIZONTAL TAB]	41	29	)
10	A	[LINE FEED]	42	2A	*
11	B	[VERTICAL TAB]	43	2B	+
12	C	[FORM FEED]	44	2C	,
13	D	[CARRIAGE RETURN]	45	2D	-
14	E	[SHIFT OUT]	46	2E	.

Encodings, exercise 3

They look the same but there are different!

BOM = Byte Order Mark

without BOM

```
> hexdump -C 3A.txt
0000000 d1 8f d0 b7 d1 8b d0 ba 0a
```

dos2unix

with BOM

```
> hexdump -C 3B.txt
0000000 ef bb bf d1 8f d0 b7 d1 8b d0 ba 0a
```

Wikipedia: *The Unicode Standard permits the BOM in UTF-8, but does not require or recommend its use.*

Encodings, exercise 4

```
 > cat 4A.txt
é
 > cat 4B.txt
é
 > diff 4A.txt 4B.txt
1c1
< é
---
> é
```

```
> hexdump -C 4A.txt
0000000 c3 a9 0a
> hexdump -C 4B.txt
0000000 65 cc 81 0a
```

```
c3 a9 -> U+00E9 -> é
```

precomposed character

```
65    -> U+0065 -> e
cc 81 -> U+0301 -> ´
```

Combining character

Encodings, exercise 4

https://withblue.ink/2019/03/11/why-you-need-to-normalize-unicode-strings.html

Normalizing strings

Thankfully, there's an easy solution, which is **normalizing the string** into the "canonical form".

There are four standard normalization forms:

- `NFC` : Normalization Form Canonical Composition
- `NFD` : Normalization Form Canonical Decomposition
- `NFKC` : Normalization Form Compatibility Composition
- `NFKD` : Normalization Form Compatibility Decomposition

The most common one is `NFC` , which means that first all characters are decomposed, and then all combining sequences are re-composed in a specific order as defined by the standard. You can choose whatever form you'd like, as long as you're consistent, so the same input always leads to the same result.

https://docs.python.org/3/library/unicodedata.html#unicodedata.normalize

Encodings, exercise 5

Different characters that are almost the same!

```
> hexdump -C 5A.txt
0000000 54 6f 6d 27 73 20 63 61 74 0a
```

```
> hexdump -C 5B.txt
0000000 54 6f 6d e2 80 99 73 20 63 61 74 0a
```

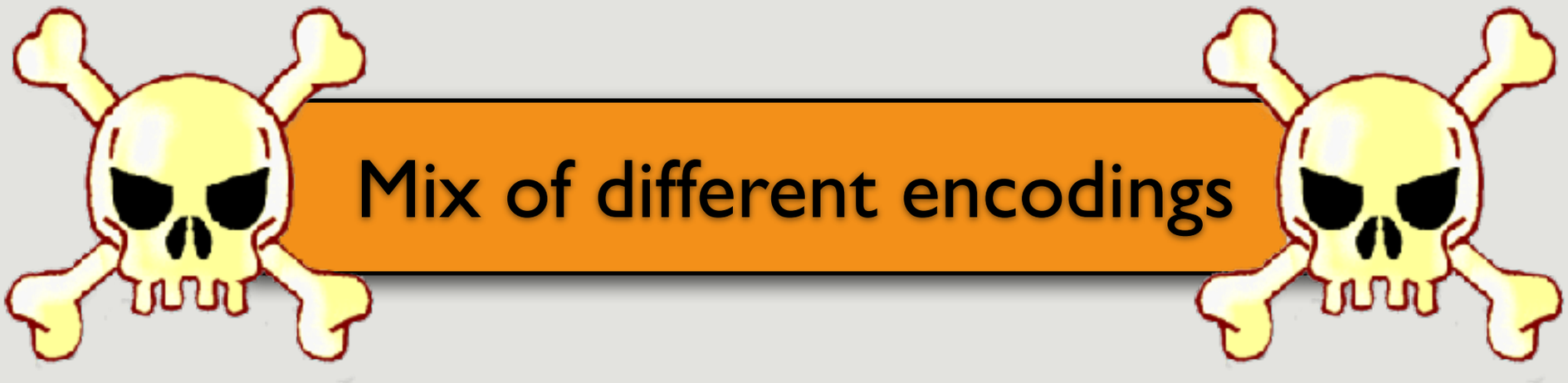
```
> hexdump -C 5C.txt
0000000 54 6f 6d ca bc 73 20 63 61 74 0a
```

nom	glyphe	Unicode
Apostrophe	Oo'Oo	U+0027 = 39
Guillemet-apostrophe	Oo’Oo	U+2019 = 8217
Lettre modificative apostrophe	Oo’Oo	U+02BC = 700

More on Wikipedia (fr)

Conclusion

1. You must always use **UTF-8** (unless you have a good reason)!
2. There is **no good reason** not to use UTF-8!
3. But you may **re-use data** with another encoding
4. What is **worse** than using an encoding which is not UTF-8?



https://imsdb.com/scripts/Les-Tontons-Flingueurs.html