Natural Language Processing (NLP)

2 — Caractères, alphabets, grammaires, expressions régulières

Georges-André Silber
https://mines.paris/nlp

Session 2024/2025 — ES3A_MES-07

École des mines de Paris

Caractères et alphabets

Données sources du NLP : caractères et alphabets

MINES PARIS | PSL

Pourquoi s'intéresser aux caractères ?

- Donnée de base du NLP : caractère ∈ alphabet
- Qualité des données primordiale pour le NLP
- En 2024, le Mojibake existe toujours
- Diversité des caractères dans les langues humaines
- Traitements plus compliqués quand on ne traite pas de l'anglais

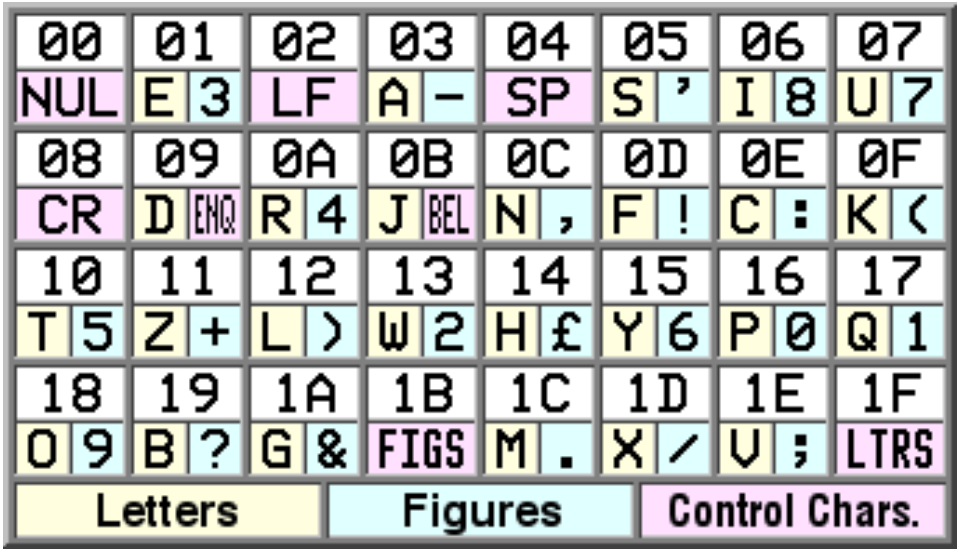
Représentation des caractères

- Sur un ordinateur, un caractère est un entier positif.
- Cet entier positif représente un caractère dans une table donnée.
- Si on ne connaît pas la table, on ne peut pas connaître le caractère.
- 1 caractère : 5 bits, 7 bits, 1, 2, 3 ou 4 octets (Unicode).

Émile Baudot (1845–1903) : code Baudot

- Code sur 5 bits (1878)
- Téléscripteur, Telex
- Baud (Bd)
  - unité de modulation
  - nb de symboles par *s*
- *Stateful*

Code Baudot (US TTY)

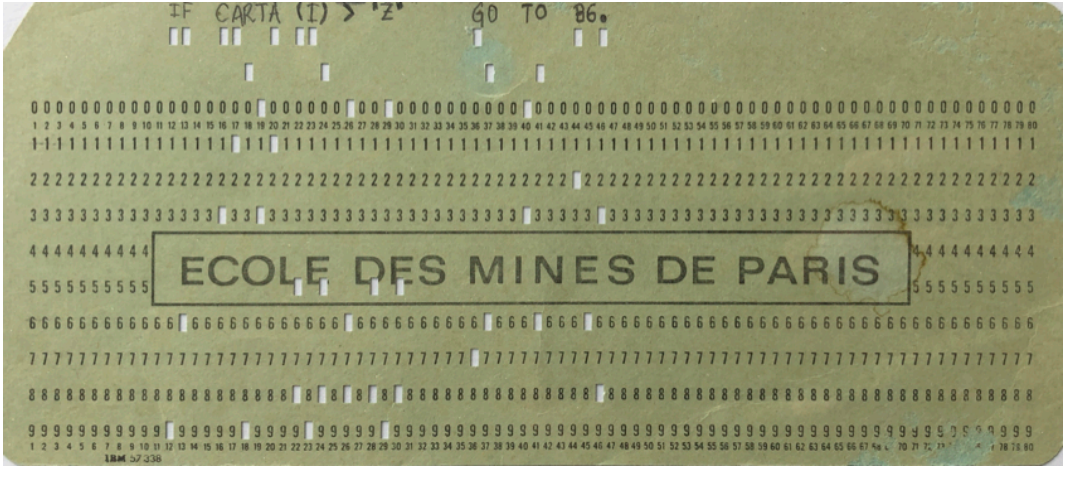


00	01	02	03	04	05	06	07
NUL	E 3	LF	A –	SP	S ’	I 8	U 7
08	09	0A	0B	0C	0D	0E	0F
CR	D ENQ	R 4	J BEL	N ,	F !	C :	K (
10	11	12	13	14	15	16	17
T 5	Z +	L)	W 2	H £	Y 6	P 0	Q 1
18	19	1A	1B	1C	1D	1E	1F
O 9	B ?	G &	FIGS	M .	X /	V ;	LTRS

Letters | Figures | Control Chars.

Lab : Code Baudot

```
$ python encode_hello.py
***.**
* *.
    . *
*  .*
*  .*
** .
  *.
*  .**
[...]
```



Extended Binary Coded Decimal Interchange Code (8 bits)

Code EBCDIC

```
$ python fortran.py
I -> 0xc9
F -> 0xc6
  -> 0x40
C -> 0xc3
A -> 0xc1
R -> 0xd9
T -> 0xe3
A -> 0xc1
[...]
```

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓ b_3 ↓ b_2 ↓ b_1 ↓ Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

American Standard Code for Information Interchange (7 bits)

Code ASCII

```
#include <stdio.h>
int main(int ac, char *av[])
{
        printf("hell\x08o\n");
}
```

Caractères étendus : ISO 8859

- Ajouts aux 96 caractères affichables de l'US-ASCII
- Encodage sur 8 bits
- 128 nouveaux caractères (utilisation du bit restant d'un octet)
- 16 parties différentes (de 8859-1 à 8859-16)
- Pas utilisable pour certaines langues d'Asie (CJCV)

Code ISO-8859-1

```
(nlp) cervinia:latin1 gasilber$ cat latin1.py
s="Hé ôh"
with open("doc.txt", "w", encoding="iso-8859-1") as f:
    f.write(s)
(nlp) cervinia:latin1 gasilber$ python latin1.py
(nlp) cervinia:latin1 gasilber$ hexdump -C doc.txt
00000000  48 e9 20 f4 68                                    |H. .
00000005
(nlp) cervinia:latin1 gasilber$ cat doc.txt
H? ?h
```

Caractères pour le monde entier : Unicode

- Développé depuis 1991 (première version).
- Standard Unicode 13 (mars 2020) : 144 697 caractères.
- Relié à la norme ISO/CEI 10646 (un sous-ensemble du standard Unicode).
- Un caractère ISO/CEI 10646 : couple nom unique / numéro unique (point de code).
- 245 000 points de codes assignés dans un espace pouvant contenir 1 114 112 codes différents (21 bits).
- 17 zones de 65 536 points de code : plans de code.
- Chaque plan de code est divisé en 4096 colonnes de code de 16 points de code.
- Codes : de `0` à `0x10FFFF` (1 114 112 - 1).
- De `0x0` à `0xFF` : ISO/CEI 8859-1.

Unicode

```
(nlp) cervinia:unicode gasilber$ python wat.py
ê == ê -> False
(nlp) cervinia:unicode gasilber$ python unicode.py
--> Aéêê 5
0 0041 Lu LATIN CAPITAL LETTER A
1 00e9 Ll LATIN SMALL LETTER E WITH ACUTE
2 00ea Ll LATIN SMALL LETTER E WITH CIRCUMFLEX
3 0065 Ll LATIN SMALL LETTER E
4 0302 Mn COMBINING CIRCUMFLEX ACCENT
(nlp) cervinia:unicode gasilber$ python spaces.py
0x2000 '\u2000' EN QUAD Zs ' '
0x2001 '\u2001' EM QUAD Zs ' '
0x2002 '\u2002' EN SPACE Zs ' '
0x2003 '\u2003' EM SPACE Zs ' '
```

UTF-8

- Universal Character Set Transformation Format sur 8 bits.
- Encodage de ISO/CEI 10646 sur 8 bits, compatible avec l'US-ASCII (`0x0` à `0x7F`).
- Stockage d'un point de code sur 1 à 4 octets consécutifs.
- Le décodage des *strings* devient *stateful*.
- Souvent compatible avec le code existant, insensible à l'*endianness*.

Définition du nombre d'octets utilisés dans le codage (attention ce tableau de principe contient des séquences invalides)

Caractères codés	Représentation binaire UTF-8	Premier octet valide (hexadécimal)	Signification
U+0000 à U+007F	0bbb·bbbb	00 à 7F	1 octet, codant jusqu'à 7 bits
U+0080 à U+07FF	110b·bbbb 10bb·bbbb	C2 à DF	2 octets, codant jusqu'à 11 bits
U+0800 à U+FFFF	1110·bbbb 10bb·bbbb 10bb·bbbb	E0 à EF	3 octets, codant jusqu'à 16 bits
U+10000 à U+10FFFF	1111·0bbb 10bb·bbbb 10bb·bbbb 10bb·bbbb	F0 à F3	4 octets, codant jusqu'à 21 bits
	1111·0100 1000·bbbb 10bb·bbbb 10bb·bbbb	F4	

UTF-8

```
(nlp) cervinia:utf8 gasilber$ python count.py
...
0 feff Cf ZERO WIDTH NO-BREAK SPACE
1 0048 Lu LATIN CAPITAL LETTER H
2 00e9 Ll LATIN SMALL LETTER E WITH ACUTE
(nlp) cervinia:utf8 gasilber$ wc -c Classeur1.csv
      XX Classeur1.csv
```

UTF-16 et UTF-32

- UTF-16 : codage sur 2 ou 4 octets (16 ou 32 bits).
- UTF-32 : codage fixe sur 4 octets (32 bits).
- Codages sensibles à l'*endianness*.
- Utilisation d'un BOM, `0xFFFE` (*little endian*) ou `0xFEFF` (*big endian*).
- Dans certains fichiers UTF-8 (par ex. CSV généré par Excel) utilisation d'un BOM pour indiquer que le fichier est UTF-8.
- Il existe également un format UTF-5 (unicode sur téléscripteur).
- Bush hid the facts

Formats de fichiers

- Format texte
- Comma-Separated Values (CSV)
- JavaScript Object Notation (JSON)
- YAML Ain't Markup Language
- TOML Tom Obvious, Minimal Language
- eXtensible Markup Language (XML)
- Portable Document Format (PDF)
  - Langage de description de page (Turing complet)
  - $b^2 - 4ac \rightarrow$ `b b mul 4 a mul c mul sub`

Langages formels et NLP

Informatique et langages

- Informatique = science du traitement automatique des données.
- Données = suite d'informations représentées dans un certain langage.
- Théorie des langages formels : étude des structures internes formelles des langages (niveau syntaxique).
- Auparavant, les systèmes d'analyse syntaxique utilisaient principalement des grammaires formelles, aidées éventuellement par des statistiques.
- Aujourd'hui, dans ces systèmes, les grammaires formelles sont complétées, voire remplacées, par des techniques d'apprentissage automatique.

Langage formels, Hiérarchie de Chomsky

Hiérarchie des grammaires formelles de N. Chomsky et M. P. Schützenberger :

- les grammaires de type 3 génèrent la famille des *langages rationnels* ou *langages réguliers*. Langages reconnaissables par les *automates finis* ;
- les grammaires de type 2 génèrent la famille des *langages algébriques*. Langages reconnaissables par les *automates à pile* ;
- les grammaires de type 1 génèrent la famille des *langages contextuels*. Langages reconnaissables par les *automates linéairement bornés* ;
- les grammaires de type 0, dites grammaires générales, génèrent la famille des *langages récursivement énumérables*. Langages reconnaissables par une *machine de Turing*.

Langages rationnels

- A : ensemble fini de symboles (*alphabet*) ;
- A^* : ensemble des mots que l'on peut former avec A ;
- Partie de A^* : un *langage* ;
- Opérations rationnelles, avec X et Y deux parties de A^* :
  - Concaténation : XY

$$\{ab, c\}\{ba, c\} = \{abba, abc, cba, cc\}$$

  - Union : $X \cup Y$

$$\{ab, c\} \cup \{ba, c\} = \{ab, ba, c\}$$

  - Étoile de Kleene (notée X^*) : le plus petit langage contenant ϵ, X et qui est clos pour l'opération de concaténation.

$$\{a, ab\}^* = \{\epsilon, a, aa, ab, aaa, ababaa, ...\}$$

Application pratique : **expressions régulières** par génération d'un automate fini (Ken Thompson). `lex`, analyseur lexical.

Grammaires algébriques

- Grammaires non-contextuelles / hors-contexte
- Règles de la forme $X \to \alpha$ où α est un terminal ou un non terminal
- Un langage est algébrique si il ∃ une grammaire algébrique le décrivant
- La plupart des langages de programmation ont une grammaire algébrique
- Peuvent être décrits sous la forme Backus-Naur (BNF)[1]
- Exemples de langages algébriques (et non rationnels) :

$$S \to aSb \mid \epsilon$$

$$\{a^n b^n \mid n \geq 0\}$$

$$S \to x \mid y \mid z \mid S + S \mid S - S \mid S * S \mid S/S \mid (S)$$

- Outils : YACC (LALR), ANTLR (LL)

1. https://fr.wikipedia.org/wiki/Forme_de_Backus-Naur

Grammaires algébriques (2)

- Panini, un grammairien de l'Inde antique a formalisé une grammaire du Sanskrit (IVe siècle avant J.C.), avec près de 4000 dans une notation proche de la BNF.
- Langages de Dyck : ensemble des mots bien parenthésés sur un ensemble fini de parenthèses ouvrantes et fermantes.

Définition (Morphisme alphabétique)
$h : A^* \to B^*$ avec A et B des monoïdes libres et h un morphisme. h est *alphabétique* si l'image d'une lettre de A est une lettre de B ou ϵ.

Théorème (Chomsky–Schützenberger)
Un langage L est algébrique ssi il existe un langage de Dyck D, un langage rationnel K et un morphisme alphabétique h tels que

$$L = h(D \cap K)$$

Grammaires algébriques (3)

- Soit la grammaire :

$$S \rightarrow S + S \mid a \mid 1$$

- Dérivations de $1 + a + a$

Dérivation gauche	Dérivation droite
S	S
$S + S$	$S + S$
$1 + S$	$S + a$
$1 + S + S$	$S + S + a$
$1 + a + S$	$S + a + a$
$1 + a + a$	$1 + a + a$

⇔ Machine de Turing non déterministe linéairement bornée (avec n la taille de l'entrée, ruban de taille kn où k est une constante indépendante de n).

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

La machine de Turing

Objet mathématique abstrait composé :

- d'une bande infinie découpé en cases pouvant contenir un symbole ;
- d'une tête de lecture pouvant à chaque étape lire un symbole, écrire un symbole, puis se déplacer sur la bande d'une case à gauche ou à droite ;
- un registre fini d'états dans lesquels peut se trouver la machine ;
- une table d'action indiquant pour un état et un symbole l'action à effectuer.

La machine de Turing

Une machine de Turing déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
- $\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction partielle de *transition*. Si δ n'est pas définie sur l'état courant et le symbole courant, la machine s'arrête ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si elle s'arrête dans un état de F.

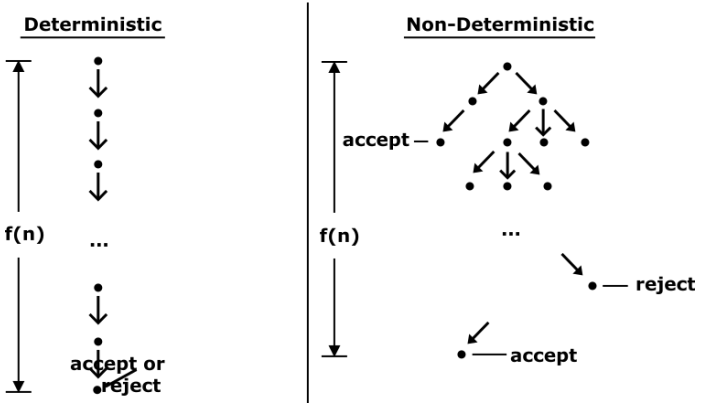
Exemple de partie de δ : $\delta(q_1, x) = (q_2, y, \leftarrow)$ indique que dans l'état q_1 quand x est lu sur la bande, on passe en état q_2, on écrit y et on se déplace à $\leftarrow$.

La machine de Turing non déterministe

Une machine de Turing non déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
- $\delta \subseteq (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ est la relation de *transition* ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si *une branche* s'arrête dans un état de F.

Machine de Turing déterministe vs non-déterministe



Expressions régulières

Expressions régulières

- Expressions régulières par génération d'un automate fini (Ken Thompson).
- `grep`, `lex`, analyseur lexical
- `https://regexcrossword.com`
- Python : `import re`
- hyperscan

Exemple 1 : utilisation dans un IDE

```
#define MAX_URI_COUNTRY 3
#define MAX_URI_CORPUS 5
#define MAX_URI_NATURE 70
#define MAX_URI_YEAR 5
#define MAX_URI_MONTH 3
#define MAX_URI_DAY 3
#define MAX_URI_NUMBER 30
#define MAX_URI_VERSION 9
#define MAX_URI 256

MAX_(\w+)
$1_MAX
```

Exemple 2 : découpage d'un arrêt de cour d'appel d'Agen

```
intro_re = re.compile(
  r'^(?P<intro>.*?)(?=('
  r'<p>\s*A\s+rendu\s+l.arrêt\s+((réputé\s+)?
  r'contradictoire|par\s+défaut)'
  r'|<p>\s*EXPOS(É|E)\s*DU\s*LITIGE'
  r'|<p>A rendu réputé l.arrêt réputé contradictoire'
  r'))',
  re.UNICODE|re.DOTALL|re.MULTILINE|re.IGNORECASE)
decision_re = re.compile(
  r'(?P<decision>(<p>par\s*ces\s*motifs).*)$',
  re.U|re.DOTALL|re.MULTILINE|re.IGNORECASE)
```

Exemple 3 : numéros d'alinéas (Droit Quotidien)

```
alinea_number = (
  r"("
  r"\w\)(?=\s+)"
  r"|\d{1,2}°(\s+bis)?(?=\.?\s+)"
  r"|\d{1,2}(\s+bis)?(?=\.?\s+)"
  r"|[IVX]+(?=\.\s+)"
  r")"
)
```