

Texte : Cours

Représentation d'un texte en machine

La mémoire ne contient que des 0 et des 1, ainsi lorsqu'on voit une lettre à l'écran, en mémoire c'est *un entier* qui *encode* ce caractère.

Différents encodages existent, nous allons les étudier.

Un caractère ?

Comment enregistrer, de manière optimale, du texte en mémoire ?

De combien de symboles a-t-on besoin ?

- 26 lettres dans l'alphabet, 52 avec les majuscules.
- 10 chiffres 0123456789
- Un peu de ponctuation : , ; : ! ? . / * \$ - + = () [] { } " ' etc.
- Quelques caractères techniques (retour à la ligne, espace etc.)

On dépasse $2^6 = 64$ mais en se contentant du minimum, on reste en dessous de $2^7 = 128$.

L'idée est de trouver une représentation de chacun de ces caractères par un nombre. C'est un *encodage*.

On peut encoder une table assez vaste avec 7 bits.

Idée d'ASCII (1961) : uniformiser les nombreux encodages incompatibles entre eux.

L'encodage ASCII est l'encodage universel dont tous les autres dérivent.

La table ASCII complète

Remarques sur la table précédente

- Elle se lit ainsi :) est encodé par l'entier $41_{10} = 29_{16}$.
- Tout élément de la table est codé sur 7 bits, 1 octet par caractère suffit
- Les caractères codés entre 32 et 127 sont imprimables, les autres sont des caractères de contrôle.
- Certains caractères de contrôle ont un effet sur le texte comme le numéro 9 : \t (tabulation) ou le 10 : \n (retour à la ligne).
- Les chiffres commencent à 30_{16} , les majuscules à 41_{16} et les minuscules à 61_{16}
- Pour obtenir la notation binaire, on part de l'hexa.
Premier chiffre : 3 bits, second chiffre 4 bits

$$A \rightarrow 41_{16} \rightarrow 4 \times 16 + 1 \rightarrow 100\ 0001$$

ou

$$s \rightarrow 73_{16} \rightarrow 7 \times 16 + 3 \rightarrow 111\ 0011$$

ASCII TABLE

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]

Figure 1: La table ASCII

Les 95 caractères imprimables

- Seulement 95 caractères imprimables, pas de caractère accentués :

hex	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x2.		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5.	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x6.	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7.	p	q	r	s	t	u	v	w	x	y	z	{			}	~

Question : Expliquer la phrase suivante : “Pour changer la casse d’un caractère, il suffit de changer 1 bit dans sa représentation ASCII.”

Résumé ASCII

En ASCII on encode un caractère sur 7 bits.

Cet encodage ne comporte que 95 caractères imprimables (de 20 à 127) et aucun accent.

C’est le seul encodage “qui marche toujours”.

Si vous voyez un accent, ce n’est pas de l’ASCII.

L’organisation astucieuse permet de passer facilement de passer des majuscules aux minuscules.

Python et la table ascii

Les fonctions `chr` et `ord` permettent d’accéder à la table

```
>>> chr(65) # caractère 65 (décimal)
'A'
>>> ord('A') # numéro décimal du caractère
65
```

Question : Comment reproduire la table ci-dessous en quelques lignes ?

```
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

Réponse

```
for c in range(32, 128):
    if c and (c % 32) == 0:
        print()
    print(chr(c), end=" ")
print()
```

Explications

Pour `c` allant de 32 à 127 inclus, - si 32 divise `c`, aller à la ligne : `print()` - afficher `chr(c)` qui est le caractère de la table ASCII ¹

¹ `chr` utilise l’encodage UTF-8 qui étend ASCII, on en parlera plus tard.

iso-8859-1 ou iso-Latin-1

Comment compléter la table ASCII ?

Première idée : conserver une taille fixe de 1 octet. 256 octets en tout dont 128 dans la table ASCII, il en reste 128 libres.

L'encodage iso-8859-1, dit iso-Latin-1 est apparu en 1986 et est utilisé en Europe de l'ouest. D'autres versions pour les caractères iso-Latin-2 de l'Europe de l'est etc.

- Reprend la table ascii et ajoute les accents au coût d'un bit supplémentaire (à gauche).
- Ainsi ASCII(A) = 0100 0001 = iso-latin1(A).
- Chaque caractère d'une table iso-xxxx-x occupe exactement 1 octet.
- Les caractères "spéciaux" commencent donc à 128 jusque 255.
- Encore incomplet : œ et Æ n'y sont pas ! *Ce qui a contribué à leur disparition de nombreux documents écrits dans les années 90...*
- Windows (Windows-1252) et Mac (MacRoman) ont leurs propres versions rendant l'échange de documents et développement de logiciels **plus que pénibles**.
- Les variantes iso-Latin-1, iso-Latin-2 etc sont incompatibles entre-elles.

Bref, c'est ~~de la merde~~ imparfait mais cet encodage étant encore la norme sous Windows, il faudra faire avec.

Unicode

L'unicode et en particulier **UTF-8** vise à résoudre TOUS les problèmes dans UNE norme.

- minimiser l'espace occupé par un caractère
- proposer un encodage adaptable à tous les caractères employés sur terre
- conserver l'ordre de la table ascii de départ

Unicode remonte à 1991, est encore en développement, comporte déjà 137 374 caractères d'une centaine d'écritures dont les idéogrammes, l'alphabet grec etc.

UTF-8 est utilisé par 90,5% des sites web en 2017 et dans la majorité des systèmes UNIX (comprenez les serveurs et appareils mobiles).

Motivation d'unicode : \$ et £

Les machines des années 1980 étant fournies avec leur propre encodage, une somme d'argent en dollars se voyait attribuer le symbole monétaire \$ aux USA et le symbole £ au Royaume Uni (symbole monétaire de la livre sterling).

Mais entre dollars (\$) et livres (£) et les confusions étaient fréquentes.

Bob est aux USA et saisit un message pour Alice, au Royaume Uni :

Bob ---> Alice

> Je t'envoie 100\$

Sur son écran Alice voit apparaître :

Alice <--- Bob

> Je t'envoie 100£

La somme *réellement transférée est 100\$* mais Alice *croit recevoir 100£* dont le montant est différent...

On a ensuite, peu à peu, étendu ce projet à tous les symboles existant.

Principe simplifié d'UTF-8

- UTF-x utilise une taille variable pour chaque caractère. Les plus courants occupent 1 bits.
- Chaque caractère est codé avec une séquence de 1 à 4 octets.
- Un texte encodé en ASCII est encodé de la même manière en UTF8 (sauf exception)
- Les premiers bits indiquent la taille de la séquence :
 - 0xxxxxxx : 1 octet
 - 110xxxxx 10xxxxxxxx : 2 octets

- 1110xxxx 10xxxxxx 10xxxxxx : 3 octets
- 11110xxx 1001xxxx 10xxxxxx 10xxxxxx : 4 octets

- Lorsqu'un document est encodé en UTF-8 :
 - Si un caractère commence par un 0 alors il occupe 1 octet et c'est le même que dans la table ASCII : 0100 0001 -> A
 - Sinon on compte le nombre 1 initiaux. Par exemple 1110... alors le caractère occupe 3 octets. Il faut lire aussi les deux octets suivants pour déterminer le caractère complet.
- On note U+XXXX un caractère encodé en UTF-8
- La taille est variable (génant pour les développeurs novices), l'espace en mémoire est parfois important
- Un caractère peut avoir plusieurs représentations → problèmes de sécurité informatique : certaines opérations interdites sont filtrées en reconnaissant des caractères. Ce problème est globalement résolu.
- Lorsqu'un caractère UTF-8 nécessite plusieurs octets aucun n'est un code ASCII valide. Aucune confusion n'est possible.

Table complète

Table UTF-8 complète

Python et l'UTF-8

Python 2 (l'ancien) et l'encodage des caractères

Python 2 supporte bien UTF-8 à condition de lui demander.

Sans quoi le premier accent va faire planter python 2.

On trouve souvent dans l'entête d'un fichier .py :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

qui signifient :

- Exécute ce fichier avec **python**, situé dans le dossier **/usr/bin/env**
- l'encodage du fichier est en **utf-8**

Python 3 (le notre)

Python 3 supporte nativement utf-8, on peut se passer de cette précision On utilise les fonctions **chr** et **ord**

- **chr(entier)** retourne le caractère encodé par cet entier en utf-8
- **ord(caractère)** retourne l'encodage utf-8 de ce caractère.

Martine écrit en UTF-8

WHAT ?

- La lettre **é** du mot "écrit" a été *encodée* en **UTF-8** (parce que 2 caractères sont affichés)
En mémoire elle occupe 2 octets (elle n'est pas dans la table ascii)
- Ces deux octets ont été *décodés* en **iso-latin1** (1 octet par caractère), respectivement en Æ et ©

Complément : base64

Contexte

L'email ne supporte que le texte. Afin de transmettre autre chose que du texte par email (images, vidéos, sons, dossiers compressés etc.) il convient de représenter le contenu du document sous la forme d'un texte.

GILBERT DELAHAYE - MARCEL MARLIER

martine

Ã©crit en UTF-8



casterman

Figure 2: martine

Base 64

Base64 est l'encodage utilisé pour transmettre des pièces jointes par email. C'est un groupe de schéma pour encoder des données binaires sous forme d'un texte au format ASCII grâce à la représentation de ces données en base 64. Le terme base64 vient à l'origine de l'encodage utilisé pour transférer certains contenus MIME (Extensions multifonctions du courrier Internet).

base64 est aussi employé pour transmettre du contenu dans les URL.

Principes de base 64

Cet encodage utilise 65 symboles de la table ASCII pour encoder 6 bits par un caractère.

Ce processus consiste à encoder 24 bits par une chaîne de 4 caractères.

On procède de gauche à droite, en concaténant 3 octets pour créer un seul groupement de 24 bits (8 bits par octet). Ils sont alors séparés en 4 nombres de seulement 6 bits ($2^6 = 64$ d'où le nom). Chacune des 4 valeurs est enfin représentée par un caractère de l'alphabet retenu.

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage				
0	000000	A	17	010001	R	34	100010	i	51	110011	z
1	000001	B	18	010010	S	35	100011	j	52	110100	0
2	000010	C	19	010011	T	36	100100	k	53	110101	1
3	000011	D	20	010100	U	37	100101	l	54	110110	2
4	000100	E	21	010101	V	38	100110	m	55	110111	3
5	000101	F	22	010110	W	39	100111	n	56	111000	4
6	000110	G	23	010111	X	40	101000	o	57	111001	5
7	000111	H	24	011000	Y	41	101001	p	58	111010	6
8	001000	I	25	011001	Z	42	101010	q	59	111011	7
9	001001	J	26	011010	a	43	101011	r	60	111100	8
10	001010	K	27	011011	b	44	101100	s	61	111101	9
11	001011	L	28	011100	c	45	101101	t	62	111110	+
12	001100	M	29	011101	d	46	101110	u	63	111111	/
13	001101	N	30	011110	e	47	101111	v			
14	001110	O	31	011111	f	48	110000	w	(complément) =		
15	001111	P	32	100000	g	49	110001	x			
16	010000	Q	33	100001	h	50	110010	y			

La description complète est disponible [ici](#)

Remarques

- Ce codage augmente la taille des données : la taille des données est augmentée d'au moins un tiers. Les caractères "blancs" (espace, tabulation, retour à la ligne) augmentent encore plus la taille.

Avec ce codage, même les caractères lisibles dans les données d'origine sont encodés de manière illisible.

- L'intérêt de l'encodage base64 ne se trouve donc pas dans la représentation de données textuelles, mais surtout dans la représentation de données binaires.

Lorsque l'on veut représenter des données binaires (une image, un exécutable) dans un document textuel, tel qu'un courriel, la transcription hexadécimale en ASCII des octets multiplierait la taille par deux, l'encodage en base64 permet de limiter cette augmentation.

Par ailleurs, le reproche fait sur la lisibilité des données tombe de lui-même dans ces conditions : les données binaires n'ont pas vocation à être compréhensibles sans interprétation par un logiciel dédié (cas d'une image, par exemple).

Exemple

Prenons le groupe de 3 caractères ASCII "Hi!". Ci-dessous la première ligne indique en binaire l'équivalence de ces 3 octets. La transformation consiste comme on peut le voir, à séparer les bits pour former en sortie 4 groupes de 6 bits :

```
8 bits  8 bits  8 bits  <=> 6 bits 6 bits 6 bits 6 bits
  H      i      !
01001000 01101001 00100001 <=> 010010 000110 100100 100001
```

S G k h

Les 4 groupes de 6 bits en sortie nous donnent les valeurs 18, 6, 36 et 33. Ainsi en suivant la correspondance de la table indexée nous obtenons les 4 caractères “SGkh”

Avec la commande base64 sous Linux :

```
$ echo -n 'Hi!' | base64
```

```
SGkh
```

Et pour décoder :

```
$ echo -n 'SGkh' | base64 -d
```

```
Hi!
```

```
>>> import base64
>>> base64.b64encode(b'Hi!')
b'SGkh'
>>> base64.b64decode(b'SGkh')
b'Hi!'
>>>
```

Si nous prenons une chaîne de caractères qui n’a pas un nombre de bits multiple de 24, on complète avec un ou deux symboles =. Par exemple la chaîne “Salut” :

```
S        a        l        u        t
01010011 01100001 01101100 01110101 01110100
010100 110110 000101 101100 011101 010111 010000 ?????? (nombre de bits multiple de 24)
U        2        F        s        d        X        Q        =
```