RND. Texte

I/ Introduction

Comme vous le savez peut-être, un ordinateur ne peut stocker que des nombres, ou plus précisément des 0 et des 1 (des « *bits* ») qu'on regroupe pour former des nombres en binaire. Comment fait-on alors pour écrire du texte ? La réponse est simple : on associe à chaque **caractère** (une lettre, un signe de ponctuation, une espace…) un **nombre**. Un texte est alors une suite de ces nombres, on parle de **chaîne de caractères**.

II/ La norme ASCII

Avant 1960 de nombreux systèmes de codage de caractères existaient, ils étaient souvent incompatibles entre eux.

En 1960, l'organisation internationale de normalisation (ISO) décide de mettre un peu d'ordre dans ce bazar en créant la norme ASCII (American Standard Code for Information Interchange). À chaque caractère est associé un nombre binaire.

Le jeu de caractères ASCII utilise **7 bits** (et non 8 !) et dispose donc de **128 (27) caractères uniquement, numérotés de 0 à 127**. En effet, il est paru à une époque où des regroupements par 7 au lieu de 8 étaient encore assez fréquents.

Voici le tableau ASCII

Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char
0x00	0	NULL	Null	0x20	32	Space		0x40	64	@		0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!		0x41	65	A		0x61	97	a
0x02	2	STX	Start of text	0x22	34	"		0x42	66	B		0x62	98	b
0x03	3	ETX	End of text	0x23	35	#		0x43	67	C		0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$		0x44	68	D		0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%		0x45	69	E		0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&		0x46	70	F		0x66	102	f
0x07	7	BELL	Bell	0x27	39	'		0x47	71	G		0x67	103	g
0x08	8	BS	Backspace	0x28	40	(		0x48	72	H		0x68	104	h
0x09	9	TAB	Horisontal tab	0x29	41	)		0x49	73	I		0x69	105	i
0x0A	10	LF	New line	0x2A	42	*		0x4A	74	J		0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+		0x4B	75	K		0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,		0x4C	76	L		0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-		0x4D	77	M		0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.		0x4E	78	N		0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/		0x4F	79	O		0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0		0x50	80	P		0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1		0x51	81	Q		0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2		0x52	82	R		0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3		0x53	83	S		0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4		0x54	84	T		0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5		0x55	85	U		0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6		0x56	86	V		0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7		0x57	87	W		0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8		0x58	88	X		0x78	120	x
0x19	25	EM	End of medium	0x39	57	9		0x59	89	Y		0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:		0x5A	90	Z		0x7A	122	z
0x1B	27	ESC	Escape	0x3B	59	;		0x5B	91	[		0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<		0x5C	92	\		0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=		0x5D	93	]		0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>		0x5E	94	^		0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?		0x5F	95	_		0x7F	127	DEL

On retrouve dans ce tableau :

- les 26 lettres de l'alphabet latin en minuscules et majuscules,
- la représentation des chiffres,
- des caractères types « espace », « retour à la ligne » (des touches « clavier ») qui sont aussi des caractères.

La répartition des codes ne s'est pas faite au hasard, on remarque plusieurs choses :

- il existe un intervalle de 32 entre une lettre en majuscule et son équivalent en minuscule. Ainsi, « A » est codé 65_{16} et « a » 97_{16},
- le caractère « NULL » est codé 00_{16},
- le caractère « DEL » est codé FF16.

A l'époque où la norme ASCII a été mise en place, on communiquait encore parfois à l'ordinateur à l'aide de cartes perforées.

Chaque emplacement codait un bit : 1 s'il y avait un trou, 0 sinon. La perforation était irréversible. Lorsqu'on n'avait pas encore spécifié de caractère particulier, on laissait tous les emplacements intacts et le caractère valait donc 0 (tous les bits à 0). Ce caractère « non spécifié » se retrouve en ASCII avec NULL, le caractère nul, qui vaut 0. De même, lorsqu'on voulait effacer un caractère on perçait tous les emplacements, ce qui donnait 127 (tous les bits à 1) et le caractère ASCII DEL (*delete*) correspond justement à cette suppression.

III/ La norme ISO-8859-1

La **norme ASCII** *(en anglais, American Standard Code for Information Interchange)* convient bien à la langue anglaise, mais pose des problèmes dans d'autres langues, par exemple le français. En effet **l'ASCII ne prévoit pas d'encoder les lettres accentuées**. C'est pour répondre à ce problème qu'est née la norme **ISO-8859-1**. Cette norme reprend les mêmes principes que l'ASCII, mais les nombres binaires associés à chaque caractère sont codés sur 8 bits, ce qui permet d'encoder jusqu'à 256 caractères. Cette norme va être principalement utilisée dans les pays européens puisqu'elle permet d'encoder les caractères utilisés dans les principales langues européennes (la norme ISO-8859-1 est aussi appelée "latin-1" car elle permet d'encoder les caractères de l'alphabet dit "latin").

A noter : la **norme latin-1** est compatible avec la **norme ASCII**.

Problème, il existe beaucoup d'autres langues dans le monde qui n'utilisent pas l'alphabet dit "latin", par exemple le chinois ou le japonais ! D'autres normes ont donc dû voir le jour, par exemple la norme "GB2312" pour le chinois simplifié ou encore la norme "JIS_X_0208" pour le japonais.

Cette **multiplication des normes** a très rapidement posé problème puisque bien sûr, elles sont **incompatibles** entre elles. Impossible d'ouvrir un document écrit en arabe si on ne dispose pas de la bonne norme !

IV/ La norme Unicode

Afin de régler définitivement le problème d'encodage des caractères, une norme est apparue au début des années 90 : **Unicode**. Cette table de caractère contient plus de 135 000 symboles, l'objectif étant qu'elle couvre tous les besoins inimaginables. Environ 17% des codes disponibles ont été actuellement pourvus, ce qui laisse beaucoup de marge.

On ne peut toutefois l'utiliser directement : cela multiplierait par 3 la taille d'un fichier pour un même contenu puisque chaque caractère est codé sur 3 octets au lieu d'1 seul.

Il existe donc plusieurs codages plus économes comme **UTF-8**. Cette table reprend la **norme ASCII** pour les 127 premiers codes et une séquence particulière permet d'accéder aux autres caractères de la table, qui sont codés entre 2 et 4 octets. Il s'agit donc d'un codage à taille variable, ce qui économise de la mémoire.

Unicode et l'encodage UTF-8 sont la solution actuelle aux problèmes d'encodage de texte. Depuis la version 3, Python représente ses chaînes de caractères en **UTF-8**.

A noter : **Il est très recommandé d'utiliser l'encodage UTF-8 lors de la programmation de pages Web.**

Un résumé ici sur la vidéo : https://www.youtube.com/watch?v=q_F4ffWPUzE
Auteur : Cours Python 3 , durée : 6 min 34 sec.