CODER UN CARACTÈRE EN BINAIRE

Pour coder en binaire une chaîne de caractère, ou même un texte entier, on utilise traditionnellement une « page de code », c'est-à-dire un grand tableau qui associe à chacun des caractères que l'on compte utiliser, un numéro unique.

Ces pages de code se sont progressivement standardisées pour permettre les échanges entre ordinateurs de marques différentes, puis de pays différents.

Nous retiendrons ci-dessous trois étapes majeures de cette standardisation.

I) La table ASCII

Dans les années 1960, naît une première initiative encore aujourd'hui incontournable : La table ASCII (American Standard Code for Information Interchange).

Cette table utilise 7 bits et permets donc de coder 128 caractères différents :

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

Remarques :

- Les codes ASCII vont de 0x00 à 0x7F.
  Par exemple le code 0x21 correspond au point d'exclamation.
- Pour passer des majuscules aux minuscules et vice-versa, il suffit d'ajouter ou de retrancher 0x20.
- Pour coder un chiffre, il suffit d'ajouter 0x30 à la valeur du chiffre.
- Le code 0x20 correspond à l'espace blanc.
- Les deux premières lignes correspondent à des caractères non imprimables « de contrôle » dont la plupart ne sont plus utilisés de nos jours.

Exercice :

La fonction `chr(i)` de Python permet d'obtenir le caractère de code `i`.
En tapant les commandes ci-dessous dans la console Python, essayer de deviner le rôle des caractères de contrôle correspondants :

Commande	Caractère de contrôle	Signification
`print("toto"+chr(8)+"a")`	BS	
`print("toto"+chr(9)+"a")`	HT	
`print("toto"+chr(10)+"a")`	LF	
`print("toto"+chr(13)+"a")`	CR	

II) La norme ISO 8859-1

La table ASCII convient bien aux anglophones mais ne contient aucun caractère accentué.

Aussi, quand les processeurs 8 bits se sont généralisés dans les années 70-80, on a cherché à tirer profit du 8ème bit non utilisé par ASCII pour ajouter des caractères supplémentaires.

Ainsi est née en 1986 la norme « ISO 8859-1 » souvent appelée « latin-1 » et qui a été très utilisée par les pays occidentaux.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
A	Nbsp	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Remarques :

- Cette nouvelle table reprend intégralement la table ASCII.
- Le fait d'ajouter un seul bit permet de doubler le nombre de caractères disponibles.
- Il y a toujours un écart de 0x20 entre les majuscules et les minuscules accentuées.
- Il manque le caractère € qui n'existait pas encore à l'époque.

Exercice :

Une adresse mémoire de mon ordinateur contient 1110 1010.
À quoi peut correspondre cette valeur dans les cas suivants :

- C'est un entier non signé ? ☐
- C'est un entier signé sur 8 bits ? ☐
- C'est un caractère encodé en latin-1 ? ☐

III) La norme Unicode

La table ISO 8859-1 convenait bien pour les américains et les européens de l'ouest mais ne permettait pas à une personne utilisant par exemple une version de Windows française de lire directement un texte grec ou russe.

Avec l'avènement d'Internet et la possibilité d'atteindre en un clic des sites web écrits dans d'autres alphabets, une nouvelle norme appelée « Unicode » a donc été créée dès les années 90 avec pour objectif de pouvoir encoder tous les alphabets et symboles connus.

Le début de la table Unicode sur Wikipedia

Remarques :

- Dans la table Unicode, chaque caractère a un numéro appelé « point de code » compris entre 0x0 et 0x10FFFF, soit plus d'un million de possibilités !
- Actuellement, environ 150000 caractères sont définis. Il y a donc encore beaucoup de place disponible pour de nouveaux caractères ou symboles.
- La table Unicode commence en reprenant intégralement la table latin-1 (qui elle même commence en reprenant la table ASCII).
- Python 3 utilise par défaut Unicode
- Les fonctions `chr()` et `ord()` de Python permettent d'obtenir le point de code d'un caractère Unicode et réciproquement.

Encodage UTF-8 :

En latin-1 et ASCII, un caractère prend un octet en mémoire. En revanche, si l'on choisit d'encoder un caractère Unicode par son point de code, il prendra 3 octets en mémoire !

On préfère donc utiliser « l'UTF-8 » qui est une astuce permettant de représenter **tous** les caractères Unicode en utilisant de 1 à 4 octets selon les caractères (un octet pour le début de la table, c'est à dire pour les caractères les plus utilisés et jusqu'à 4 octets pour la fin de la table, c'est à dire les caractères les moins utilisés).

Pour encoder un caractère en UTF8, on regarde donc combien de bits sont nécessaires pour convertir son point de code en binaire :

Nombre de bits	Codage en UTF8
0 → 7	0XXX XXXX
8 → 11	110X XXXX – 10XX XXXX
12 → 16	1110 XXXX – 10XX XXXX – 10XX XXXX
17 → 21	1111 0XXX – 10XX XXXX – 10XX XXXX – 10XX XXXX

Exemples :

Caractère	Point de code : hex → bin	#bits	Codage UTF8
A	41 → 0100 0001	7	0100 0001
à	E0 → 1110 0000	8	1100 0011 1010 0000
‰	2030 → 0010 0000 0011 0000	14	1110 0010 1000 0000 1011 0000