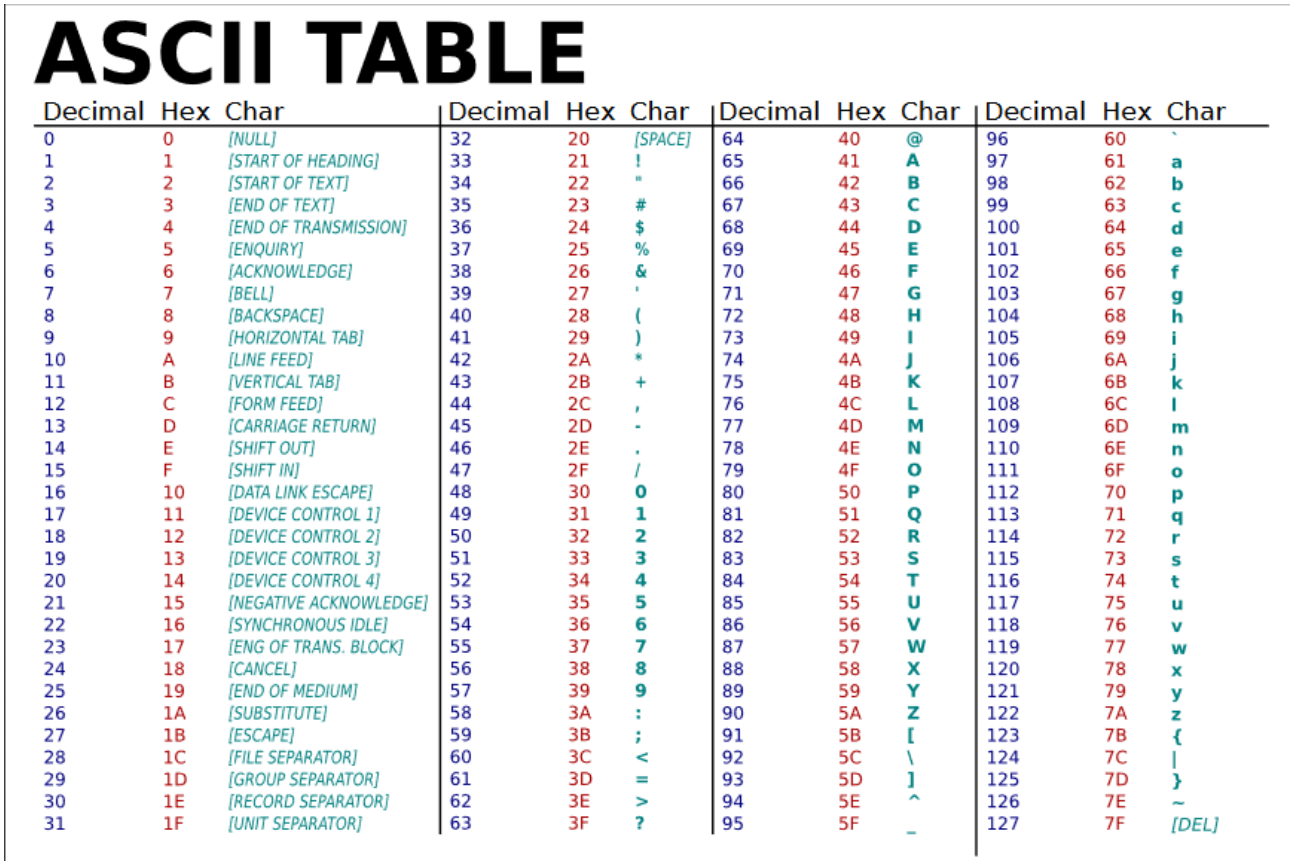
Codage des caractères

Objectifs :

Comprendre comment un ordinateur qui ne manipule que des 0 et des 1 fait pour coder un caractère.
Comprendre pourquoi les premiers codages ont dû évoluer.
Comprendre et différencier UTF-8, Unicode et ASCII.

I La première norme

Avant 1960 de nombreux systèmes de codage de caractères existaient, ils étaient souvent incompatibles entre eux. À partir de 1960, l'International Organization for Standardization (ISO) l'organisation internationale de normalisation décide de mettre un peu d'ordre en créant la norme ASCII (American Standard Code for Information Interchange) dont la version finale a été publiée en 1986. À chaque caractère est associé un nombre binaire sur un octet. Seuls les 7 premiers bits sont utilisés pour coder un caractère, le 8e bit n'est pas utilisé. Avec 7 bits il est possible de coder jusqu'à 128 caractères ce qui est suffisant pour un texte écrit en langue anglaise (pas d'accents ni d'autres lettres particulières).



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]

Comme vous pouvez le constater dans le tableau ci-dessus, au "A" majuscule correspond 65 en base 10, 41_{16} en base 16 ou encore $0100\ 0001_2$ en binaire sur un octet.

On peut constater que 32 premiers caractères et le dernier sont des codes systèmes et ne servent pas à coder des caractères, ils correspondent à des commandes console.

Limitation : Cette norme ne permet pas d'écrire d'autres langues que l'anglais correctement. Ce qui a poussé à développer d'autres normes qui permettent d'écrire correctement tous les caractères possibles y compris les langues non latines.

Influence : Cette norme a eu une forte influence dans le monde informatique. C'est pourquoi jusqu'il y a peu les noms de domaine, les adresses e-mails, les noms dans les systèmes de fichiers et les caractères utilisables dans un programme informatique étaient limités à ceux contenus dans la table ASCII. Cela permettait d'assurer l'échange de textes sans erreur.

II ISO-8859-1

L'ASCII est permet d'écrire uniquement l'anglais. Pour répondre à ce problème est née la norme ISO-8859-1 qui code aussi les caractères sur un octet. Cette norme reprend le l'ASCII pour les nombres de 0 à 127 et utilise les nombres de 128 à 255 pour coder d'autres 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 « latin1 » car elle permet d'encoder les caractères de l'alphabet dit « latin1 ». Cette norme et ses évolutions jusqu'à la norme ISO-8859-15 qui permet de coder de plus le caractère « € » a connu un certain succès au début d'Internet jusque dans les années 2010.

Dans les pays aux langues non latines comme la Chine ou le Japon d'autres normes ont été créées. Ces normes étant incompatibles entre elles, afficher ou échanger des e-mails dans des codages différents était très compliqué.

Le développement d'Internet d'abord par l'échange d'e-maisl entre universitaires de pays aux langues éloignées puis pour tout un chacun (sites internationaux, groupes internationaux, etc.) ont fait émerger très vite le besoin d'une norme internationale pour toutes les langues. C'est pour cette raison que l'industrie informatique a formé un consortium pour créer la norme Unicode

III Unicode et UTF-8

Unicode a pour ambition de rassembler tous les caractères existant afin qu'une personne utilisant Unicode puisse, sans changer la configuration de son traitement de texte, à la fois lire des textes en français ou en japonais.

Unicode est uniquement une table qui tend à regrouper tous les caractères existants au monde, il ne s'occupe pas de la façon dont les caractères sont codés dans la machine. Unicode accepte plusieurs systèmes de codage : UTF-8, UTF-16, UTF-32. Le plus utilisé est UTF-8 (Universal character set Transformation Format). Par exemple, plus de 80% des serveurs Web utilisent UTF-8.

Pour encoder les caractères Unicode, UTF-8 utilise de un à quatre octets : les caractères correspondant à ceux de la table ASCII sont codés sur un octet, alors que les autres sont codés sur un nombre d'octets plus important. On n'entrera pas dans le détail de ce codage.

Un des avantages d'UTF-8 c'est qu'il est totalement compatible avec la norme ASCII : Les caractères Unicode codés avec UTF-8 ont exactement le même code que les mêmes caractères en ASCII.

Un autre avantage est que le poids du fichier est plus léger. En effet, si tous les caractères étaient codés sur quatre octets un texte écrit en anglais que l'on code sur un octet en UTF 8 le serait sur 4 fois plus d'octets et seraient donc 4 fois plus gros. Ce qui est à la fois mieux pour le stockage et pour la transmission de textes.

Par contre, il rend plus compliqué des opérations sur les chaînes. Par exemple, pour connaître le nombre de caractères d'une chaîne, il ne suffit pas compter le nombre d'octets, il faut lire tous les octets et déterminer s'ils codent un caractère ou qu'il faut d'autres octets pour coder un caractère.

Pour se rendre compte de la diversité des symboles présents dans la norme Unicode:

http://www.unicode.org/charts/

UTF-8 est en train de devenir le standard de l'encodage des caractères, par exemple, dans cet entête d'une page du site unicode.org, on trouve bien que l'encodage est l'UTF-8.

```
<!DOCTYPE html>
<html class="avada-html-layout-wide avada-html-header-position-left avada-has-site
<head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Unicode &#8211; The World Standard for Text and Emoji</title>
```

IV En Python

Depuis la version 3 Python est un langage qui par défaut s'écrit en UTF-8 (auparavant, par défaut, il était écrit en ASCII).
La fonction **`ord`** permet d'obtenir le nombre en base 10 selon la norme UTF-8 associé à un caractère.

Exemple

```
>>> ord('A')
65
>>> ord('€')
8364
```

On peut remarquer que 'A' est bien codé sur un octet car 65<128 et '€' est codé sur deux octets.
La fonction **`chr`** permet d'obtenir le caractère UTF-8 associé à un nombre en base 10.

On peut aussi l'afficher en utilisant \u suivi de la valeur hexadécimale du caractère.

Exemple

```
>>> chr(65)
'A'
>>> chr(8364)
'€'
>>> chr(128516)
'😄'
>>> "\u26a0"
'⚠'
```