Intérêt de l'hexadécimal

I – ASCII

Les textes sont in fine stockés sous forme de 0 et de 1, comme les entiers ou les flottants.

Il existe donc des tables de correspondance. Espace → 32, A → 65, B → 66, a-→ 97 …

Mais un texte, c'est plus que des caractères. Il a aussi les caractères « invisibles », dits de contrôle.

- Début du texte (start of text STX) → 3
- Fin du texte (end of text ETX) → 4
- Les tabulations TAB → 9
- Les passages à la ligne (retour chariot RC, RC en anglais *carriage return*) → 13

La table ASCII est le standard des **128** premiers caractères (de 0 à 127 uniquement).

Elle est publiée pour la première fois en 1963. Voir l'annexe.

Exercice 01 : Trouver le texte encodé par **80 101 114 99 101 118 97 108 32 63**

Comme on a besoin de 128 cases, les ranger dans un tableau de 10 sur 13, c'est pas pratique.

On préfère faire du rangement en 8 sur 16, soit 128 cases. Pas de trou.

II – Hexadécimal

2.1 - Codification des nombres

Les _____ **chiffres** de la base 16 sont :

On notera que $\mathbf{A}_{16} = \mathbf{10}_{10}$ $\mathbf{B}_{16} = \mathbf{11}_{10}$ $\mathbf{C}_{16} = \mathbf{12}_{10}$ $\mathbf{D}_{16} = \mathbf{13}_{10}$ $\mathbf{E}_{16} = {}_4\mathbf{14}_{10}$ $\mathrm{F}_{16} = \mathbf{15}_{10}$

On peut donc représenter un **nombre** comme un ensemble de cases contenant un chiffre.

La case de poids faible est la plus à droite. Elle code la valeur _____ qu'on peut écrire _____

A chaque fois qu'on se déplace vers la gauche, la case code _____ fois plus que la précédente.

La case code	4096	256	16	1
La case code	16^3	16^2	16^1	16^0
Exemple N =	**5**	**2**	**D**	**A**
On obtient	5 x 4096	2 x 256	13 x 16	10 x 1

Au total, le nombre vaut : $\mathbf{N = 52DA}_{16} = (5*4096 + 2*256 + 13*16 + 10)_{10} = \mathbf{21210}_{10}$

Exercices 02/03/04 : Déterminer les valeurs suivantes en décimal : $\mathbf{41}_{16}$, $\mathbf{FF}_{16}$, $\mathbf{FE}_{16}$

2.2 – Augmentation d'une unité

Pour cela, il suffit de placer le prochain chiffre dans la liste

`0 ⇨ 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 5 ⇨ 6 ⇨ 7 ⇨ 8 ⇨ 9 ⇨ A ⇨ B ⇨ C ⇨ D ⇨ E⇨ F`

- Si on est déja au dernier chiffre (F), on revient au premier chiffre (0) dans cette case ET on rajoute UN dans la case juste à gauche.

Exercice de cours : Quel est le nombre suivant **en base 16 ?** On travaillera avec

`E +1 =` `F + 1 =` `13 + 1 =` `F3 +1 =` `FD + 1 =`

`FF +1 =` `3F + 1 =`

2.3 – Nombre de cas dénombrables

Si on dispose de X cases pour écrire un nombre en base 10, on dispose de $\mathbf{16^X}$ valeurs possibles.

Attention, la valeur minimale étant le cas 0, la valeur maximale est $\mathbf{16^X - 1}$

Exercice de cours : Trouver le nombre de cas et la valeur maximale avec 2 chiffres ou 3 chiffres en base 16.

III – Binaire vers Hexadécimal : utilisation du quartet

Perceval reçoit en réalité :

`0101000001100101011100100110001101100101011101100110000101101100001000000111111`

Si on fait des tas de 8 bits, ca donne :

`01010000` `01100101` **`01110010`** `01100011` **`01100101`** `01110110` **`01100001`** `01101100` **`00100000`** `00111111`

Exercices 06 : Convertir les deux premiers octets en décimal pour voir si le début du message est bien $80_{10}\,101_{10}$

Il est plus simple d'utiliser les quartets : 4 bits à la fois, on s'en sort une fois qu'on a l'habitude.

`0000` donne $\mathbf{0}_{16}$	`0100` donne $\mathbf{4}_{16}$	`1000` donne $\mathbf{8}_{16}$	`1100` donne $\mathbf{C}_{16}$ (12_{10})
`0001` donne $\mathbf{1}_{16}$	`0101` donne $\mathbf{5}_{16}$	`1001` donne $\mathbf{9}_{16}$	`1101` donne $\mathbf{D}_{16}$ (13_{10})
`0010` donne $\mathbf{2}_{16}$	`0110` donne $\mathbf{6}_{16}$	`1010` donne $\mathbf{A}_{16}$	`1110` donne $\mathbf{E}_{16}$ (14_{10})
`0011` donne $\mathbf{3}_{16}$	`0111` donne $\mathbf{7}_{16}$	`1011` donne $\mathbf{B}_{16}$	`1111` donne $\mathbf{F}_{16}$ (15_{0})

L'octet **`01010000`** se découpe en deux quartets **`0101 0000`** qu'on transforme en hexadécimal **`5 0`** $_{16}$

Exercices 07: Vérifier que les octets soient bien équivalent à `65 72 63` en hexadécimal.

`0110 0101` `0111 0010` `0110 0011`

Exercices 08: Décoder le début du message texte en utilisant la table Hexa ci-dessous.

`5 0 6 5 7 2 6 3 6 5 7 6 6 4 6 C 2 0 3 F`

Exercices 09: Vérifier que $\mathbf{50}_{16}$ corresponde bien à $\mathbf{80}_{10}$. Vérifier que les deux tables donnent bien le caractère **`P`**.

IV – Hexadécimal et couleurs

Les couleurs sont créées sur les écrans en activant trois sous-pixels RGB.

Pas d'activation :	0 %	**`0000 0000`** $_2$	ou	**0** $_{10}$	ou	$\mathbf{0}_{16}$
Pleine activation	100 %	**`1111 1111`** $_2$	ou	**255** $_{10}$	ou	$\mathbf{FF}_{16}$

Exemple : `#FF45F8` veut directement

Exercice de cours : Ecrire les codes couleurs hexadécimaux possibles des couleurs suivantes en HTML :

Rouge : **Vert foncée :** **Jaune :**

V – Hexadécimal en Python

Préfixes permettant de préciser la base des nombres

Pour fournir un nombre en binaire, il fallait le précéder de `0b`. **Pour l'hexadécimal, c'est `0x`.**

On utilise cette notation (par exemple **`0xFF`**) car c'est plus facile que d'afficher les indices (par exemple $\mathbf{FF}_{16}$).

```
>>> 0xFF
255
>>> 0xFF + 1
256
>>> 1 + 0xF 16
16
>>> 0b11111111 + 0xFF
510
```

Pour convertir un string représentant un hexa en décimal : `>>> int('0xFF', 16)` renvoie **`255`**

Pour convertir un nombre en hexadécimal, on utilise la fonction **hex** :

```
>>> hex(255)
'0xff'
>>> hex(15)
'0xf'
>>> hex(0b1111)
'0xf'
>>> hex(0b1111+1)
'0x10'
```

Si vous ne voulez pas les deux premiers caractères **`0x`** indiquant hexadécimal, il ne faut prendre que les caractères d'index 2 et plus :

```
>>> hex(255)[2:]
```

VI – Encodage un octet, Unicode et UTF-8

Limitation de l'ASCII : uniquement 128 caractères (0-127), il manque la plupart des caractères spéciaux ou accentuées.

Solution : L'ASCII étendue : les tables 1 octet :

→ Chaque zone géographique avait sa propre table d'un octet qui comportait 256 caractères (0-255).
→ Il faut indiquer la table d'encodage utilisée pour créer le fichier texte qui n'est en fait qu'une suite de nombres.
→ Il faut du coup également connaître la table utilisée pour parvenir à décoder correctement les nombres !

Problème avec l'apparition d'Internet :
→ Apparition de documents sur lequel on a besoin de caractères situées sur des tables différentes
→ Difficile de trouver la table par défaut si le fichier ne contient pas d'indication de l'encodage utilisée pour créer le document

Solution UNICODE : un nombre pour un caractère
→ Il ne s'agit pas d'une table d'encodage réelle mais d'une simple association d'un nombre avec un caractère

Utilisation UNICODE : la façon la plus usuelle d'utiliser UNICODE est via l'UTF, un processus d'encodage qui transforme le nombre UNICODE en suite de bits. Vu le nombre de caractères différents, il faudrait 4 octets pour parvenir à associer naturellement un nombre UNICODE à une suite de bits.

UTF-8 : Table d'encodage utilisant 1 octet pour les caractères usuels et 2 ou 4 octets pour les autres.
UTF-16 : Table d'encodage utilisant 2 octets pour les caractères usuels et 2 ou 4 octets pour les autres.
UTF-32 : Table d'encodage utilisant 4 octets pour tous caractères (fichiers lourds)

TABLE ASCII

0	NUL	*Null* (nul)
1	SOH	*Start of Heading* (début d'en-tête)
2	STX	*Start of Text* (début de texte)
3	ETX	*End of Text* (fin de texte)
4	EOT	*End of Transmission* (fin de transmission)
5	ENQ	*Enquiry* (demande)
6	ACK	*Acknowledge* (accusé de réception)
7	BEL	*Bell* (sonnerie)
8	BS	*Backspace* (espacement arrière)
9	HT	*Horizontal Tab* (tabulation horizontale)
10	LF	*Line Feed* (saut de ligne)
11	VT	*Vertical Tab* (tabulation verticale)
12	FF	*Form Feed* (saut de page)
13	CR	*Carriage Return* (retour chariot/retour à la ligne)
14	SO	*Shift Out* (code spécial)
15	SI	*Shift In* (code standard)
16	DLE	*Data Link Escape* (échappement en transmission)
17	DC1	*Device Control 1* à *4* (contrôle de périphérique)
18	DC2	
19	DC3	
20	DC4	
21	NAK	*Negative Acknowledge* (accusé de réception négatif)
22	SYN	*Synchronous Idle* (attente synchronisée)
23	ETB	*End of Transmission Block* (fin de bloc de transmission)
24	CAN	*Cancel* (annulation)
25	EM	*End of Medium* (fin de support)
26	SUB	*Substitute* (remplacement)
27	ESC	*Escape* (échappement)
28	FS	*File Separator* (séparateur de fichier)
29	GS	*Group Separator* (séparateur de groupe)
30	RS	*Record Separator* (séparateur d'enregistrement)
31	US	*Unit Separator* (séparateur d'unité)

32	**Space**	48	**0**	64	**@**	80	**P**	96	**`**	112	**p**
33	**!**	49	**1**	65	**A**	81	**Q**	97	**a**	113	**q**
34	**"**	50	**2**	66	**B**	82	**R**	98	**b**	114	**r**
35	**#**	51	**3**	67	**C**	83	**S**	99	**c**	115	**s**
36	**$**	52	**4**	68	**D**	84	**T**	100	**d**	116	**t**
37	**%**	53	**5**	69	**E**	85	**U**	101	**e**	117	**u**
38	**&**	54	**6**	70	**F**	86	**V**	102	**f**	118	**v**
39	**'**	55	**7**	71	**G**	87	**W**	103	**g**	119	**w**
40	**(**	56	**8**	72	**H**	88	**X**	104	**h**	120	**x**
41	**)**	57	**9**	73	**I**	89	**Y**	105	**i**	121	**y**
42	*****	58	**:**	74	**J**	90	**Z**	106	**j**	122	**z**
43	**+**	59	**;**	75	**K**	91	**[**	107	**k**	123	**{**
44	**,**	60	**<**	76	**L**	92	****	108	**l**	124	**|**
45	**-**	61	**=**	77	**M**	93	**]**	109	**m**	125	**}**
46	**.**	62	**>**	78	**N**	94	**^**	110	**n**	126	**~**
47	**/**	63	**?**	79	**O**	95	**_**	111	**o**	127	**DEL**

	_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	**_0**
1_	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	**1_**
2_		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	**2_**
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	**3_**
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	**4_**
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	**5_**
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	**6_**
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	**7_**
	_0	**_1**	**_2**	**_3**	**_4**	**_5**	**_6**	**_7**	**_8**	**_9**	**_A**	**_B**	**_C**	**_D**	**_E**	**_F**	

Afficher un caractère spécial en HTML : &#NUMERO; exemple : **@** pour @
Afficher un caractère spécial en Python : chr(64) pour afficher @