Le codage de l'information

1/ Introduction

Les informations utilisées en informatique sont des valeurs (nombres) ou des symboles (lettres).
Or, l'information élémentaire utilisée par les systèmes numériques est une information logique (Vrai ou Faux, Tout ou Rien, 0V ou 5V, 0 ou 1).
Il est donc nécessaire de définir une convention pour coder une information en respectant la contrainte de n'utiliser que des informations logiques élémentaires.
Python utilise un certain nombre de types de convention. Nous étudierons le codage des booléens (`bool`), des entiers ou *integer* en anglais (`int`), des nombres décimaux (`float`), et des chaines de caractères (`string`).

1.1/ *Vocabulaire :*

- Un « Bit» (contraction américaine de Binary digiT) est un digit du système binaire (valeur 0 ou 1).
- Un « Mot» est un ensemble de bits dont il faut préciser le nombre. Par exemple: un mot de 16 bits, un mot de 12 bits, ...
- Un « Quartet » est un mot de 4 bits.
- Un « Octet» est un mot de 8 bits. On dit aussi « Byte » en angais. Dans le cas de notation anglo-saxonnes,il ne faut pas confondre aussi bit (b) et Byte (B).

Depuis décembre 1998, une norme internationale a imposé que les préfixes kilo, méga, giga, téra, etc., correspondent aux mêmes multiplicateurs que dans tous les autres domaines. Ce qui donne:

- 1 kilooctet (ko ou kB) = 10^3 octets = 1 000 octets
- 1 mégaoctet (Mo ou MB) = 10^6 octets = 1 000 ko
- 1 gigaoctet (Go ou GB) = 10^9 octets = 1 000 Mo
- 1 téraoctet (To ou TB) = 10^{12} octets = 1 000 Go
- 1 pétaoctet (Po ou PB) = 10^{15} octets = 1 000 To

Mais on peut encore compter comme avant 1998. Il faut alors utiliser d'autres appellations:

- 1 kilo binaire octet , appelé kibioctet (kio) = 2^{10} octets = 1 024 octets
- 1 méga binaire octet , appelé mébioctet (Mio) = 2^{20} octets = 1 024 Kio
- 1 giga binaire octet , appelé gibioctet (Gio) = 2^{30} octets = 1 024 Mio
- 1 téra binaire octet, appelé tébioctet (Tio) = 2^{40} octets = 1 024 Gio
- 1 péta binaire octet, appelé pébioctet (Pio) = 2^{50} octets = 1 024 Tio

2/ Codage d'un nombre

On utilise aujourd'hui des systèmes de numération dits pondérés ou positionnelle.
La définition d'un système de numération pondéré repose sur trois notions:

- La« base » du système: c'est un nombre entier, noté B.
- Les « chiffres » ou « diqits » du système : ce sont des caractères, tous différents, représentant chacun un élément de la base . Il y en a donc B au total, notés 0, 1, 2, 3, 4, ... Pour écrire un nombre, on associe plusieurs chiffre dans un ordre déterminé, par exemple: N =1354 ou N =4153 ou ...
- Le« poids » de chaque chiffre selon son rang (sa position dans l'écriture). Compté de la droite vers la gauche, ce poids vaut B^0 (c'est à dire 1) pour le premier chiffre, B^1 (c'est à dire B) pour le second chiffre, B^2 pour le troisième chiffre, etc ...

On notera un nombre sous la forme : $(ijkl)_B$ où i,j,k et l sont des chiffres, et B la base du système.

2.1/ ***Exemple : Base décimale, ou système décimale***

Dans ce système, la base B est 10. Il y a 10 chiffres notés : 0, 1, 2, 3,4, 5, 6, 7, 8, 9
Les nombres 3997 et 195,28 exprimés en décimal signifient :

- $3997 = 3x10^3 + 9x10^2 + 9x10^1 + 7x10^0$
- $195{,}28 = 1x10^2 + 9x10^1 + 5x10^0 + 2x10^{-1} + 8x10^{-2}$

En base 10, on ne note pas $(3997)_{10}$ ou $(195{,}28)_{10}$, mais simplement 3997 ou 195,28.

2.2/ ***Sens de lecture***

Rappelons que ces chiffres indiens nous sont parvenus par le monde arabo-musulman où on écrit de la droite vers la gauche. En partant de la droite, on peut lire aisément un nombre, chiffre après chiffre: 7 unités, et 9 dizaines, et 9 centaines, et 3 milliers.

La lecture de gauche à droite, à la mode occidentale, est impossible sans une vision d'ensemble du nombre.
En effet, on lit 3, puis trente neuf, puis trois cent quatre vingt dix neuf, avant de comprendre que c'est en fait trois mille neuf cent quatre vingt dix sept.

On comprend alors la convention : dans un mot, (un octet par exemple), le « Bit de poids faible» (L.S.B. en américain, Less Significant Bit) est le bit situé le plus à droite, donc de plus faible poids.
Le « Bit de poids fort» (M.S.B. en américain, Most Significant Bit) est le bit situé le plus à gauche.

2.3/ ***Système binaire***

Le mathématicien et philosophe allemand Leibniz (1646-1716) a été un des premiers à étudier la numération binaire. Mais le système binaire est réellement utilisé depuis le XIXème siècle et les travaux du mathématicien anglais George Boole (nous étudierons plus tard l'algèbre de Boole). C'est aujourd'hui le système qui permet de traiter et représenter les informations par ordinateur.
Dans ce système, la base B est 2. Il y a 2 chiffres notés: 0 et 1
Les nombres $(1101)_2$ et $(101{,}01)_2$ exprimés en binaire signifient :
$(1101)_2 = 1x2^0 + 0x2^1 + 1x2^2 + 1x2^3 = 1x1 + 0x2 + 1x4 + 1x8 = 13$ (en base 10).
$(101{,}01)_2 = 1x2^0 + 0x2^1 + 1x2^2 + 0x2^{-1} + 1x2^{-2} = 1x1 + 0x2 + 1x4 + 0x0{,}5 + 1x0{,}25 = 5{,}25$ (en base 10).
Il est alors aisé de définir **l'ordre naturel** croissant des nombres binaires :

0
1
10
11
100
101

2.4/ ***Conversion***

a) Binaire → décimal

Pour un entier naturel N, codé en système binaire, $N_2 = (a_n, a_{n-1}, \dots, a_0)_2$

L'expression générale est N_{10} s'écrit

$$N_{10} = \sum_{i=0}^{n} a_i . 2^i$$

(N_2 et N_{10} sont des expressions différentes de la même quantité N)

b) Décimal → Binaire

La conversion inverse peut être facilement effectuée par récurrence.

Application 1 : Exprimer 243 en système binaire.

2.5/ *Codage des entiers naturels et relatifs*

Sur un octet (8 bits), on peut coder les entiers naturels de 0 à 255. Sur 2 octets (16 bits), on peut coder les entiers naturels de 0 à 65535. De manière générale, sur n bits on peut coder les entiers naturels de 0 à $2^n - 1$.

Application 2 :

a) Quel est le plus grand entier naturel représentable dans un mot d'un octet ?
b) Combien de bits sont nécessaires pour coder 999 ?
c) Justifier que : $a + b + c = a + c + b$

Si l'on souhaite préciser le signe, un bit peut être consacré à cette information (celui de gauche dans le mot) :

- 0 pour indiquer un entier positif ;
- 1 pour indiquer un entier négatif.

Avec un mot d'un octet (8 bits), il est donc possible de représenter un entier de -127 à +127.

Un inconvénient de cette méthode est que 0 peut être codé de deux manières (+0 et -0). Mais surtout, l'algorithme d'addition des entiers naturels n'est plus adapté.

Application 3 : Justifier que l'algorithme d'addition n'est plus fonctionnel avec ce codage.

Pour éviter cette lacune, en informatique industrielle (langage C, par exemple), les entiers relatifs sont codés selon la notation en **complément à deux**. Pour un mot de $n+1$ bits, le bit le plus à gauche est associé au poids (-2^n), les autres respectant la convention de codage des entiers naturels. Si ce bit est à 1, le nombre est forcément négatif :

Ce qui permet de représenter les entiers de -2^n à $2^n - 1$

Par exemple, pour un codage sur un octet, il est possible de représenter les entiers naturels de -128 à 127 :

$$N = -2^7.a_7 + \sum_{i=0}^{6} a_i.2^i$$

$$-128 = -2^7.1 + 0 = (10000000)_{\text{bns}}$$
$$-127 = -2^7.1 + 1 = (10000001)_{\text{bns}}$$
$$0 = -2^7.0 + 0 = (00000000)_{\text{bns}}$$
$$127 = -2^7.0 + 127 = (01111111)_{\text{bns}}$$

Dans cette configuration, il n'est donc pas possible de coder le nombre 128. Une conséquence fâcheuse est que le résultat du calcul 64+64 ne peut être correctement déterminé. On parle alors de **dépassement de capacité**.

Application 4 :

a) Python utilise 32 bits pour coder ses entiers relatifs (*int*). Dans quel intervalle Python peut-il représenter des entiers ?
b) Quel est l'expression codant -32 en entier naturel sur un octet ? Additionner ce nombre binaire avec lui-même. Décoder le résultat. Analyser la cohérence.
c) Quel est l'expression codant 64 en entier naturel sur un octet ? Additionner ce nombre binaire avec lui-même. Décoder le résultat. Analyser la cohérence.
d) Déterminer l'opposé de 64 codé en entier naturel sur un octet. Proposer une méthode générale pour déterminer l'opposé d'un nombre codé en complément à deux.

2.6/ ***Codage des « réels »***

On a vu qu'il était possible de coder des nombres à virgules en utilisant des puissances de 2 négatives :
5,25 = 1x4 + 0x2 + 1x1 + 0x0,5 + 1x0,25 = $1x2^0 + 0x2^1 + 1x2^2 + 0x2^{-1} + 1x2^{-2}$.
Il ne sera donc pas possible de coder exactement en binaire tous les réels, comme par exemple π ou $\sqrt{3}$. Coder un réel implique forcément des erreurs d'arrondis.

Le standard IEEE 754 (1985) définit un format de représentation des nombres à virgule flottante (float) en binaire. C'est ce codage qui sera présenté ici. Parmi les formats proposés, nous ne retiendrons que la représentation binaire simple précision (32 bits) et double précision (64 bits), utilisé par Python (le principe reste le même pour des mots de tailles différentes).

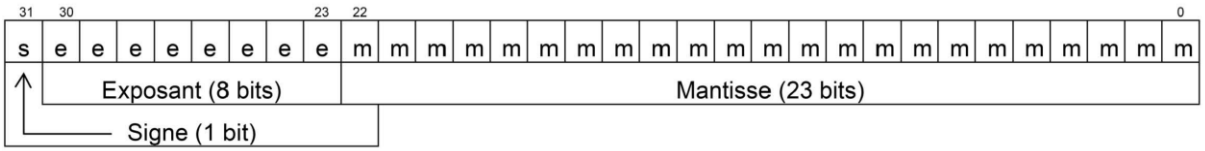
Le principe s'expose facilement en système décimal, par ce qui est appelée la notation scientifique d'un nombre x sous la forme $x = \pm m.10^e$
Où :

- $m \in [1\ ;10\ [$ est appelé la mantisse ou significande ;
- e, entier relatif, est appelé l'exposant.

En représentation binaire simple précision (sur 32 bits) : $x = \pm m.2^e$

- $m \in [1\ ;2\ [$, codé sur 23 bits : $m = 1 + \sum m_i.2^{-i}$
- e entier relatif, codé sur 8 bits : $e = \sum e_i.2^i - 127$
- un bit pour le signe

Le code simple précision va se mettre sous cette forme :



On remarque que l'exposant a été décalé de 127. C'est pour permettre les exposants négatifs. L'exposant peut ainsi varier de -127 à +128. En réalité, l'exposant est compris entre -126 et 127, les valeurs -127 et 128 étant réservées aux codages de cas particuliers.
Le seul cas particulier à connaitre est le 0, qui ne peut être codé avec la convention précédente. On considère alors que si $e = -127$ ($e_i = 0$ pour tout i), $x = 0$.

De même, en représentation binaire double précision (sur 64 bits) : $x = \pm m.2^e$

- $m \in [1\,;2\,[$, codé sur 52 bits : $m = 1 + \sum m_i.2^{-i}$
- e entier relatif, codé sur 11 bits : $e = \sum e_i.2^i - 127$
- un bit pour le signe

Application 5 :
On se propose de coder 243,25 = $(11110011{,}01)_2$ en simple précision, soit sur 32 bits.
Il nous faut donc déterminer le *signe*, la *mantisse*, et l'*exposant.*

3/ Codage des caractères

Utilisé pour les échanges en informatique, le code ASCII (American Standard Code for Information Interchange) permet de coder les 26 lettres de l'alphabet (majuscules et minuscules), les 10 chiffres et les signes de ponctuation : il utilise un octet (8 bits), ou son équivalent sous forme d'entier décimal. Il est notamment utilisé comme convention de communication entre un clavier et un PC.
Cet octet donne une certaine souplesse d'utilisation , puisqu'il permet de coder des commandes de contrôle en plus des caractères alphanumériques (bits 1 à 7), d'utiliser le bit 8 comme bit de parité ou pour définir un deuxième tableau de caractères (caractères étendus : caractères avec accent, etc.).
Python utilise les caractères compris entre 32 et 126.

Pour former une expression littérale, il suffit alors d'utiliser une liste de caractère. Sous Python, ce type est appelé *string*.

Décimal	Caractère	Commentaire	Décimal	Caractère		Décimal	Caractère	
0	NUL	(Null char.)	43	+	(plus)	86	V	
1	SOH	(Start of Header)	44	,	(comma)	87	W	
2	STX	(Start of Text)	45	-	(minus or dash)	88	X	
3	ETX	(End of Text)	46	.	(dot)	89	Y	
4	EOT	(End of Transmission)	47	/	(forward slash)	90	Z	
5	ENQ	(Enquiry)	48	0		91	[	(left opening bracket)
6	ACK	(Acknowledgment)	49	1		92	\	(back slash)
7	BEL	(Bell)	50	2		93	]	(right closing bracket)
8	BS	(Backspace)	51	3		94	^	(caret cirumflex)
9	HT	(Horizontal Tab)	52	4		95	_	(underscore)
10	LF	(Line Feed)	53	5		96	`	
11	VT	(Vertical Tab)	54	6		97	a	
12	FF	(Form Feed)	55	7		98	b	
13	CR	(Carriage Return)	56	8		99	c	
14	SO	(Shift Out)	57	9		100	d	
15	SI	(Shift In)	58	:	(colon)	101	e	
16	DLE	(Data Link Escape)	59	;	(semi-colon)	102	f	
17	DC1	(XON)(Device Control 1)	60	<	(less than sign)	103	g	
18	DC2	(Device Control 2)	61	=	(equal sign)	104	h	
19	DC3	(XOFF)(Device Control 3)	62	>	(greater than sign)	105	i	
20	DC4	(Device Control 4)	63	?	(question mark)	106	j	
21	NAK	(Negative Acknowledgement)	64	@	(AT symbol)	107	k	
22	SYN	(Synchronous Idle)	65	A		108	l	
23	ETB	(End of Trans. Block)	66	B		109	m	
24	CAN	(Cancel)	67	C		110	n	
25	EM	(End of Medium)	68	D		111	o	
26	SUB	(Substitute)	69	E		112	p	
27	ESC	(Escape)	70	F		113	q	
28	FS	(File Separator)	71	G		114	r	
29	GS	(Group Separator)	72	H		115	s	
30	RS	(Request to Send)(Record Separator)	73	I		116	t	
31	US	(Unit Separator)	74	J		117	u	
32	SP	(Space)	75	K		118	v	
33	!	(exclamation mark)	76	L		119	w	
34	"	(double quote)	77	M		120	x	
35	#	(number sign)	78	N		121	y	
36	$	(dollar sign)	79	O		122	z	
37	%	(percent)	80	P		123	{	(left opening brace)
38	&	(ampersand)	81	Q		124	|	(vertical bar)
39	'	(single quote)	82	R		125	}	(right closing brace)
40	(	(left opening parenthesis)	83	S		126	~	(tilde)
41	)	(right closing parenthesis)	84	T		127	DEL	(delete)
42	*	(asterisk)	85	U				