Chap 2. Octets

Livre p 251 – Chap 20 Représentation approximative des nombres réels

1. Unités informatiques

a) Bits

Un ordinateur travaille en binaire, un chiffre (0 ou 1) est alors appelé un bit (**bi**nary digi**t**)

b) Octets

Un regroupement de 8 bits est appelé un octet (ou byte) : de 00000000 à 11111111 (c-à-d : de 0 à 255)
Le bit le plus à droite est appelé bit de poids faible (lsb), et celui le plus à gauche, bit de poids fort (msb).

c) Mots

Selon sa puissance, un microprocesseur travaille sur des mots (plusieurs octets) plus ou moins grands…
(32 bits : mots de 4 octets, 64 bits : mots de 8 octets)

d) Multiples normalisés (Source : Wikipédia)

Il existe deux sortes de préfixes suivant si on utilise des puissances de 10 ou des puissances de 2.
Remarque : $2^{10} = 1\,024 \approx 10^3$.

Unités de octets v · d · m

Ordre de grandeur	Système international (SI)			Préfixes binaires		
	Unité	Notation	Valeur	Unité	Notation	Valeur
1	**octet**	o	1 octet	**octet**	o	1 octet
10^3	kilooctet	ko	10^3 octets	kibioctet	Kio	2^{10} octets
10^6	mégaoctet	Mo	10^6 octets	mébioctet	Mio	2^{20} octets
10^9	gigaoctet	Go	10^9 octets	gibioctet	Gio	2^{30} octets
10^{12}	téraoctet	To	10^{12} octets	tébioctet	Tio	2^{40} octets
10^{15}	pétaoctet	Po	10^{15} octets	pébioctet	Pio	2^{50} octets

e) Evolution des capacités du stockage numérique (Source : Wikipédia)

- Cartes perforées : au 18ème siècle pour les métiers à tisser, les orgues de Barbarie et les pianos mécanique, puis au début du 20ème siècle par IBM pour l'informatique (80 caractères par carte).
- Bandes magnétiques : Dans la 2ème moitié du 20ème siècle pour l'audio, la vidéo et l'informatique. (50 octets par centimètre)
- Disquettes : 8 pouces de 80 Kio, 5"¼ de 160 Kio à 1 200 Kio, puis 3"½ de 720 Kio à 2 880 Kio (Un reportage américain montrait en 2015 des disquettes 8 pouces toujours utilisées pour l'envoi de missiles nucléaires !)
- Disques optiques : CD-Rom de 650 Mio (682 Mo) ou 747 Mio (783 Mo), DVD-Rom de 4,7 Go à 8,5 Go, puis Blu-ray de 7,5 Go à 128 Go, …
- Disques durs : Quelques dizaines ou centaines de Mo au début, plusieurs To aujourd'hui…
- Microdrive (Mémoire Flash) : De 170 Mo à 8 Go
- On trouve de nombreux autres supports de stockage : Cartes mémoires (SD), clef USB, cloud…

2. Représentation des entiers en binaire

a. Entiers courts et entiers longs

Dans beaucoup de langages informatiques il existe différentes sortes de représentations des entiers
Les entiers courts sont codés sur 16 bits : de 0 à 65 535 s'ils ne sont pas signés (entiers naturels)
de – 32 768 à 32 767 s'ils sont signés (entiers relatifs)
Les entiers longs sont codés sur 32 bits : de 0 à 4 294 967 295 s'ils ne sont pas signés
de – 2 147 483 648 à 2 147 483 648 s'ils sont signés
ou sur 64 bits : de 0 à 18 446 744 073 709 551 615 s'ils ne sont pas signés
de – 9 223 372 036 854 775 808 à 9 223 372 036 854 775 808 sinon
Remarque : Cette distinction n'existe pas en Python, seule la capacité mémoire limite la taille des entiers.

b. Entiers négatifs.

Pour représenter un entier négatif, on aurait pu simplement utiliser le premier bit pour indiquer si c'est un nombre positif ou un nombre négatif, mais cela n'est pas pratique dans les calculs !
Exemple : sur 16 bits, si $00000000\ 01011100_{b2} = 92_{b10}$ et que $10000000\ 01011100_{b2} = -92_{b10}$
Alors leur somme donnerait : $10000000\ 10111000 = -184$… pas terrible !

On utilise alors le « complément à 2 » :
Pour les entiers courts par exemple (sur 16 bits)

- Les entiers positifs (de 0 à 32 767) sont codés normalement… le 1er bit est alors 0.
- Pour les entiers négatifs, on remplace chaque bit par son complément à 2 puis on ajoute 1 (Complément à 2 : 0 devient 1, et 1 devient 0)… le 1er bit est alors 1.

Remarque : La somme d'un entier et de son opposé donne bien 0 dans ce cas.
Exemples : sur 16 bits, $00000000\ 01011100_{b2} = 92_{b10}$ et $11111111\ 10100100_{b2} = -92_{b10}$
$00000111\ 10011001_{b2} = 1\,945_{b10}$ et $11111000\ 01100111_{b2} = -1\,945_{b10}$ en binaire signé.
Attention : Quel que soit la représentation utilisée pour les entiers, il y a le risque de dépassement des capacités qui peut entrainer de nombreux bugs...
C'est un dépassement d'entier dans les registres mémoire qui a ainsi provoqué l'explosion de la fusée européenne Ariane 5 lors de son vol inaugural le 4 juin 1996 !
On prévoit également un « bug de l'an 2038 » comparable au « bug de l'an 2000 » (Source : wikipedia)

3. Représentations des réels

a. Dans le système décimal

Pour représenter les nombres réels en base 10, il existe différentes notations dont la notation scientifique.
Exemples : $a = 2019$ et $b = 0{,}000\,012$ peuvent s'écrire $a = 2{,}019 \times 10^3$ et $b = 1{,}2 \times 10^{-5}$.
Remarque : Certains réels ont un développement infini, c'est le cas de $\frac{1}{3}$ ou $\sqrt{2}$ par exemple…
(C'est la différence entre l'ensemble **D** des décimaux et l'ensemble **R** des réels)

b. Dans le système binaire

Pour représenter les nombres réels en base 2, on utilise des nombres à virgule flottante (Norme IEEE754)
Exemples : $a = 92$ peut s'écrire $a = 1011100_{b2} = 1{,}0111 \times 2^6$.
$b = 0{,}343\,75$ peut s'écrire $b = 0{,}01011_{b2} = 1{,}011 \times 2^{-2}$.
Pour représenter un nombre à virgule, il existe plusieurs formats : simple précision (32 bits), double précision (64 bits), double précision étendue (80 bits).
On utilise le modèle suivant :

1 bit de signe	*e* bits d'exposant						*m* bits de mantisse				
			…						…		

Le nombre codé vaut alors : signe × (1 + mantisse) × $2^{\text{exposant – décalage}}$, où décalage = $2^{e-1} - 1$.
Remarque : en 32 bits : $e = 8$ et $m = 23$ (décalage = 127)
en 64 bits : $e = 11$ et $m = 52$.
Exemple : Sur 32 bits l'expression 0 10000000 10010001111010111000010 correspond à

- bit de signe « 0 » donc un nombre positif, signe = 1.
- exposant « 10000000 » donc $2^7 = 128$ soit après décalage : $2^{\text{exposant – décalage}} = 2^1$
- mantisse « 10010001111010111000010 » donc

$N = (1 + 1\times2^{-1} + 0\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} + 1\times2^{-5} + \ldots + 1\times2^{-22} + 0\times2^{-23}) \times 2^1 \approx 3{,}14$ à 10^{-6} près.
Attention : Un nombre aussi simple que 0,1 en décimal à une écriture infinie en binaire et donc une valeur approchée en mémoire en virgule flottante !
Les réels ayant une écriture finie en binaire sont appelés les nombres dyadiques : $N = \frac{a}{2^k}$ où $a \in \mathbf{Z}$ et $k \in \mathbf{N}$.
(Pour $a = 0.1$ si on demande 60 chiffres significatifs en Python, avec la commande : *print*("%.60f" % *a*), on obtient : $a = 0.100000000000000005551115123125782702118158340454101562500000$)