Représentation des nombres

◇ Deux points importants :

- Dans un ordinateur, toutes les données sont représentées par des nombres ;
- Ces nombres sont représentés en *binaire* (uniquement avec les chiffres 0 et 1).

En informatique, un chiffre 0 ou 1 s'appelle un *bit* (contraction de *binary digit*).

I. La représentation binaire

◇ Un nombre s'écrit en base 2 comme une somme de puissances de 2 affectées de chiffres 0 ou 1. Par exemple :

$$207 = 128 + 64 + 8 + 4 + 2 + 1 = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

et ainsi 207 s'écrit 11001111 en base 2. On note $207 = (11001111)_b$ ou schématiquement :

$128=2^7$	$64=2^6$	$32=2^5$	$16=2^4$	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$	$=$	2^7	$+$	2^6	$+$	2^3	$+$	2^2	$+$	2^1	$+$	2^0	$=$	207
1	1	0	0	1	1	1	1														

◇ On peut réaliser les opérations usuelles à partir de la représentation binaire. À droite, le calcul de la somme de 207 et $73 = (1001001)_b$ (avec des retenues : 0+0 donne 0, 1+0 et 0+1 donnent 1 et 1+1 donne 0 et on retient

$$\begin{array}{r}
 0^1 \ 1^1 \ 1 \ 0 \ 0^1 \ 1^1 \ 1^1 \ 1^1 \ 1 \\
 + \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \\
 \hline
 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0
 \end{array}$$

1). La multiplication et la division (entière) par 2 sont très faciles à réaliser en binaire (de même que la multiplication et la division par 10 sont très faciles à réaliser en base 10) :

$$\begin{array}{ll}
 207 \cdot 2 = (110011110)_b = 414 & (\text{décalage à gauche de } (11001111)_b) \\
 207 \div 2 = (1100111)_b = 103 & (\text{décalage à droite de } (11001111)_b)
 \end{array}$$

On ne détaille pas plus les autres opérations (de toutes façons tout est réalisé de manière transparente par l'ordinateur).

II. Les nombres entiers

◇ Dans de nombreux langages de programmation, les entiers sont représentés en binaire avec un nombre fixé de *bits* (typiquement 8, 16, 32, 64).

◇ À retenir : avec une représentation sur n *bits*, on peut écrire 2^n nombres différents (il y a n cases, chacune peut contenir deux valeurs : 0 ou 1). Par exemple avec 8 *bits*, on peut représenter tous les nombres de 0 à 255. Conséquence : si 207 et 73 sont représentés sur 8 *bits*, on ne peut pas faire de même pour la somme 280 : on dit qu'il y a *débordement*. Ce problème est systématique dès que l'on travaille avec un nombre de *bits* fixé à l'avance.

◇ Avec PYTHON : le nombre de *bits* utilisés pour représenter un nombre entier est adapté automatiquement au cours du calcul, il n'y a donc pas de débordement (la seule limite est la mémoire de l'ordinateur et les temps de calculs, d'autant plus longs que les nombres manipulés sont grands).

Remarque. Les entiers relatifs sont représentés par une petite astuce que l'on ne présentera pas ici. Disons simplement qu'il faut un *bit* pour stocker le signe. Par exemple si on travaille avec 8 *bits*, alors on peut représenter les nombres (positifs) de 0 à 255 et les nombres (relatifs) de -127 à 128. □

III. Les nombres flottants

◇ Les problèmes :

- Pour des applications pratiques, il faut pouvoir représenter aussi bien de très grands nombres que de très petits ;
- L'ensemble $\mathbb{R}$ est infini, un ordinateur possède une mémoire finie, on ne pourra donc représenter qu'un sous-ensemble fini de $\mathbb{R}$.

◇ La norme IEEE-754 définit les nombres *flottants*, PYTHON utilise la version *double précision* (sur 64 *bits*). Un nombre flottant x s'écrit :

$$x = \pm m \cdot 2^{e-1023}$$

Signe : 1 *bit*, *mantisse* m : 52 *bits*, *exposant* e : 11 *bits*.

◇ Cette représentation permet d'avoir 15 chiffres significatifs (en base 10). Le plus grand nombre que l'on peut représenter est $\approx 1.79 \cdot 10^{308}$ et le plus petit nombre (strictement positif) est $\approx 2.22 \cdot 10^{-308}$.

△ **Remarques.** Conséquences de la représentation (à retenir) :

- Toute opération avec les flottants comporte des arrondis et donc des imprécisions, il est donc nécessaire de prévoir un moyen de vérifier les calculs que l'on met en œuvre (connaître l'ordre de grandeur, tester sur des cas où la solution est connue, comparer les résultats fournis par deux méthodes différentes, etc.) ;
- Même l'affichage d'un nombre comporte un arrondi éventuel ;
- Tester si deux flottants x et y sont égaux n'a pas vraiment de sens, en pratique $x = y$ est remplacé par $|x - y| \leq \varepsilon$ où ε est un flottant choisi « assez petit. » □