# Chapitre 13 : Les flottants

En python, pour coder les nombres décimaux on utilise le type $float$.

## a) Principe du codage des nombres décimaux

Le codage d'un nombre décimal $x$ s'effectue à partir de la notation scientifique : $x = (-1)^S \times (1,m) \times 2^p$

Pour une architecture à 64 bits, on a, dans l'ordre :

- 1 bit $S$ est utilisé pour le codage du signe (0 pour +, 1 pour -)
- 11 bits pour l'exposant $p$
- 52 bits sont utilisés pour la mantisse constituées de 0 et de1 (en omettant le 1e chiffre qui est nécessairement 1)

## b) Codage de l'exposant $p$

C'est la valeur $e = p + 1023$ qui est stockée : $e = p + 1023$. Sur 11 bits, les valeurs de l'exposant décalé $e$ sont comprises entre [1,2046] ($e = 0$ et $e = 2047$ étant réservées pour coder $-\infty$ et $+\infty$). Ainsi $-1022 \leq p \leq 1023$ permet le codage compris entre $10^{-308}$ et $10^{308}$.

## c) Codage de la partie décimale de la mantisse

- En base décimale : $x = \frac{5}{8} = 0{,}625 = 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$
- En binaire : $x = \frac{5}{8} = \frac{2^2+2^0}{2^3} = \frac{2^2}{2^3} + \frac{1}{2^3} = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

La conversion de la partie décimale est obtenue par multiplications successives par *2* de la partie décimale du résultat précédent, l'unité obtenue correspond à un élément de la décomposition. On procède ainsi de suite jusqu'à ce qu'il n'y ait plus de partie décimale ou que le nombre de bits obtenus corresponde à la taille du mot mémoire dans lequel on stocke cette partie. En effet, tous les nombres ne possède pas un développement fini et conduisent inévitablement à des erreurs de calculs.

$$0{,}625 * 2 = \mathbf{1}{,}250$$
$$0{,}250 \times 2 = \mathbf{0}{,}5$$
$$0{,}5 \times 2 = \mathbf{1}{,}0$$
$$\text{Donc } x = (0{,}625)_{10} = (0{,}101)_2$$
$$0{,}1 * 2 = 0{,}2$$
$$0{,}2 * 2 = 0{,}4$$
$$0{,}4 * 2 = 0{,}8$$
$$0{,}8 * 2 = 1{,}6$$
$$0{,}6 * 2 = 1{,}2$$
$$x = (0{,}1)_{10} = \left(0{,}00011\ \overline{0011}\right)_2$$

## d) Exemples

➔ Exemple 1 : $x = 16{,}625$

$$16{,}625 = 1 \times 10^1 + 6 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3} = 1{,}6625 \times 10^1$$
$$16{,}625 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$
$$16{,}625 = (10000{,}101)_2 = +1{,}0000101 \times 2^4$$

Le code en machine donne alors (en omettant le 1e 1 de la mantisse):

$$0\ \ 10000000011\ \ \ 0000101\ puis\ 45\ z\acute{e}ros$$

➔ Exemple 2 :$x = 1{,}0$

$$(1{,}0)_{10} = +(1 \times 2^0 + 0 \times 2^{-1}) = (1{,}0)_2$$

Le code en machine donne alors (en omettant le 1e 1 de la mantisse):

$$0\ \ 01111111111\ puis\ \ 52\ z\acute{e}ros$$

Ici on obtient un pas de l'ordre de $2^{-52} \approx 10^{-16}$, on observe les erreurs, appelée erreurs d'arrondies suivantes :

- 1 + h =1 si h<10^-16
- (1-(1-h)) = 0 si h<10^-16
- (1-1+h) = h même si h<10^-16

➔ Exemple 3 $x = 255{,}0$

$$(255{,}0)_{10} = (1111\ 1111{,}0)_2 = +1{,}11111110 \times 2^7$$

Le code en machine donne alors (en omettant le 1e 1 de la mantisse):

$$0\ \ 10000000110\ \ \ 1111111\ \ puis\ (52-7)\ z\acute{e}ros$$

On obtient une résolution de l'ordre de $2^{(7-52)} \approx 10^{-14}$ : le pas entre deux valeurs est plus faible

Si on retient que $\log_2 10 \equiv 3$ et que $\log_{10}(2) \equiv 0{,}3$ alors si un nombre est approximé par $10^m$ alors :

- Il est codé sur $n \approx 3m$ bits
- L'exposant est $p = 3m$

La résolution sera de l'ordre de $2^{(3m-52)} \approx 10^{0.3*(3m-52)}$

```
print(1.0*10**20+200)#1e+20 car pas typique de 251
print(1.0*10**20+20000)#1.0000000000000002e+20
```

➔ Exemple 4 : $x = x_{max}$

On peut aussi connaître le plus grand float :

$$0\ \ 111111111110\ puis\ 52\ nbre\ 1 = +(1 + 2^{-1} + 2^{-2} + \cdots 2^{-52})2^{1023} \approx 2^{1023}$$
$$\approx 2^{1000} \approx (2^{10})^{100} \approx 10^{300}$$

**Commenté [AM1]:** On va évoquer la représentation des nombres réels comme :
-Les décimaux comme 0,1
-Les rationnels comme 1/3
-Les irrationnels $\sqrt{2}$ *ou* $\pi$

| Ecriture | Type |
|---|---|
| $x = 2$ | *int* |
| $x = 6//3$ | *int* |
| $x = 2.0$ | *float* |
| $x = 6/3$ | *float* |
| $x = 200 * 10^{-2}$ | *float* |
| $x = \sqrt{4}$ | *float* |

**Commenté [AM2]:** L'écriture scientifique en base 10 consiste à écrire n'importe quelle nombre $x$ sous la forme d'une mantisse comprise entre 1 et 9.999 …
$x = \pm m10^e$
En binaire $m$ n'est constitué que de 0 et 1 et est de la forme 1,00101….

**Commenté [AM3]:** Sur 32 bits :
-8 bits pour $e$ donc les exposants possibles sont compris [1,254] et pour $p$ : [−126,127].
Ce qui correspond à des valeurs extrêmes de $\pm 10^{38}$
-23 pour $m$

**Commenté [AM4]:** Comme en base 10, le premier chiffre significatif de la mantisse est non nul et ici il vaut forcément 1 : il n'est donc pas codé.

**Commenté [AM5]:** On peut se convaincre que c'est la même démarche en base 10. On parle d'écriture dyadique

**Commenté [AM6]:** Il existe les mêmes difficultés en base 10, par exemple 1/3. Cette erreur s'appelle erreur d'arrondi ou de troncature

**Commenté [AM7]:**
-

**Commenté [AM8]:** Le plus petit nombre positif non nul étant 10^-300.
Il y a donc des dépassements de capacités (overflow) possibles