

Représentation des réels

Fiche de synthèse

① LE PROBLÈME

En Python, `0.1 + 0.2` vaut `0.30000000000000004`, et `0.1 + 0.2 == 0.3` vaut `False`. Ce n'est pas une erreur de Python : c'est une conséquence de l'écriture des nombres à virgule en binaire.

② PARTIE DÉCIMALE EN BINAIRE

Après la virgule, les chiffres binaires ont pour poids les puissances négatives de 2 : $2^{-1} = 0,5$, $2^{-2} = 0,25$, $2^{-3} = 0,125$, $2^{-4} = 0,0625$.

$$110,101_2 = 4 + 2 + 0,5 + 0,125 = 6,625$$

✎ **Partie décimale vers le binaire.** On multiplie la partie décimale par 2 : la partie entière du résultat est le bit suivant. On recommence avec la nouvelle partie décimale, jusqu'à obtenir 0.

Calcul	Bit
$0,6875 \times 2 = 1,375$	1
$0,375 \times 2 = 0,75$	0
$0,75 \times 2 = 1,5$	1
$0,5 \times 2 = 1$	1

Donc $0,6875 = 0,1011_2$. Mais pour 0,1, les calculs ne s'arrêtent jamais : $0,1 = 0,0001100110011\dots_2$, le motif 0011 se répétant sans fin. L'ordinateur, qui n'a qu'un nombre fini de bits, ne stocke qu'une **valeur approchée** de 0,1.

En 1991, l'horloge d'une batterie de missiles Patriot comptait en dixièmes de seconde : l'erreur d'arrondi accumulée en cent heures, 0,34 seconde, l'a empêchée d'intercepter un missile.

③ LA NORME IEEE 754

Définition 1 (Nombre flottant) La norme IEEE 754 (1985) écrit un nombre non nul en notation scientifique binaire :

$$(-1)^S \times 1,F \times 2^e$$

avec un bit de **signe** S , une **mantisse** F , les chiffres après la virgule, et un **exposant** e , enregistré avec un **biais** : $E = e + 127$ en simple précision.

Format	Signe	Exposant	Mantisse	Chiffres
simple, 32 bits	1	8, biais 127	23	environ 7
double, 64 bits	1	11, biais 1023	52	environ 15

Le type `float` de Python est en double précision. Le biais permet d'écrire un exposant négatif sans bit de signe : $e = -3$ est enregistré $E = 124$.

✎ **Coder $-6,625$ en simple précision.**

1. Signe : le nombre est négatif, $S = 1$.
2. Binaire : $6,625 = 110,101_2$.
3. Normaliser : $110,101_2 = 1,10101_2 \times 2^2$, donc $e = 2$.
4. Exposant : $E = 2 + 127 = 129 = 10000001_2$.
5. Mantisse : les chiffres après « 1, », complétés à 23 bits : 10101 suivi de 18 zéros.

Résultat : 1 | 10000001 | 1010100000000000000000.

④ EN PYTHON

On ne compare jamais deux flottants avec `==` :

```
import math

print(round(0.1 + 0.2, 10) == 0.3)      # True
print(math.isclose(0.1 + 0.2, 0.3))    # True
```

Les erreurs d'arrondi s'accumulent : ajouter cent mille fois 0,01 en partant de 0 donne 999,9999999992356 et non 1000. Le module `decimal` calcule exactement en base 10.

Remarque 1. Pour aller plus loin. Les exposants extrêmes représentent l'infini, le dépassement de capacité, et NaN, « pas un nombre », résultat d'un calcul indéfini comme $0/0$.

✎ **À retenir.**

- Un nombre décimal fini, comme 0,1, peut avoir une écriture binaire infinie : il est arrondi.
- Un flottant s'écrit $(-1)^S \times 1,F \times 2^e$, avec un exposant biaisé.
- Python utilise la double précision, 64 bits, environ 15 chiffres significatifs.
- On compare deux flottants avec une tolérance, jamais avec `==`.