NSI · PREMIÈRE / COURS

Les nombres à virgule en machine

Astrolab Cours · astrolabcours.fr

01 Le problème

Tape `0.1 + 0.2` dans Python : il répond `0.30000000000000004`. Ce n'est pas un bug de Python : aucun ordinateur ne peut stocker exactement 0,1. Pourquoi ?

01 L'idée : la règle graduée incomplète

Une règle graduée au millimètre ne permet pas de mesurer exactement 1/3 de centimètre : on lit la graduation la plus proche. La machine a des graduations très serrées, mais en nombre fini. Tout nombre qui ne tombe pas pile sur une graduation est arrondi à la plus proche.

01 Les chiffres après la virgule en binaire

Après la virgule, les bits valent $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16} \ldots$

$$0{,}101_2 = \frac{1}{2} + \frac{1}{8} = 0{,}625$$

Certains nombres simples en base 10 ont une écriture **infinie** en base 2, comme 1/3 en base 10 ($0{,}333\ldots$). C'est le cas de 0,1 :

$$0{,}1 = 0{,}000110011001100\ldots_2$$

Comme on ne garde qu'un nombre fini de bits, 0,1 est stocké de façon approchée.

01 Les flottants

Un nombre **flottant** (type `float` en Python) s'écrit comme en notation scientifique, mais en base 2 :

$$x = \text{signe} \times 1{,}\text{mantisse} \times 2^{\text{exposant}}$$

La norme **IEEE 754** en double précision utilise 64 bits : 1 bit de signe, 11 bits d'exposant, 52 bits de mantisse. Cela donne environ 15 à 17 chiffres significatifs.

Les conséquences

- `0.1 + 0.2 == 0.3` vaut `False`.
- Les erreurs d'arrondi peuvent s'accumuler dans une longue boucle.
- Pour comparer, on teste si l'écart est très petit :

© Astrolab Cours · astrolabcours.fr/nsi/premiere/nombres-flottants/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

```
abs((0.1 + 0.2) - 0.3) < 1e-9   # True
```

Pour l'argent, on compte souvent en **centimes entiers** pour éviter tout arrondi.

Exemple guidé

Écris 0,375 en binaire, puis explique pourquoi 0,1 ne s'écrit pas exactement.

1. **0,375** : on multiplie par 2 et on garde la partie entière à chaque fois. $0{,}375 \times 2 = 0{,}75$ (0), $0{,}75 \times 2 = 1{,}5$ (1), $0{,}5 \times 2 = 1$ (1). Donc $0{,}375 = 0{,}011_2$, car $\frac{1}{4} + \frac{1}{8} = 0{,}375$.
2. **0,1** : $0{,}2$ (0), $0{,}4$ (0), $0{,}8$ (0), $1{,}6$ (1), $1{,}2$ (1), puis on retombe sur $0{,}2$: le motif 0011 se répète **à l'infini**.
3. Un flottant n'a qu'un nombre fini de bits (52 pour la mantisse en double précision) : on doit **arrondir**.

Conséquence en Python : on ne teste jamais `x == 0.3`, mais `abs(x - 0.3) < 1e-9`.

Retour au problème

0,1 a une écriture binaire infinie, comme $\frac{1}{3} = 0{,}333\ldots$ en décimal. L'ordinateur en garde une valeur approchée, et les petites erreurs de 0,1 et de 0,2 s'ajoutent.

Les pièges

- Croire que l'erreur vient de Python : tous les langages ont le même comportement.
- Tester l'égalité de deux flottants avec `==`.
- Les entiers Python (`int`) sont exacts ; seuls les `float` sont approchés.

© Astrolab Cours · astrolabcours.fr/nsi/premiere/nombres-flottants/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits