Les nombres flottants

Doc. 4,5

Objectif

Comprendre pourquoi un ordinateur ne peut écrire la plupart des nombres à virgule que de façon approchée, et en tirer une règle de programmation : ne jamais tester l'égalité de deux flottants.

Q1. Sans utiliser l'ordinateur, prévoir ce qu'affiche Python pour l'instruction `0.1 + 0.2 == 0.3`. Vérifier ensuite dans la console, puis afficher `0.1 + 0.2`.

Les nombres à virgule sont représentés en Python par le type `float`. Le mot vient de l'anglais *floating point*, « virgule flottante ». Pour comprendre ce qui se passe, il faut d'abord savoir écrire un nombre à virgule en base 2.

§1. Des chiffres après la virgule en base 2

En base 10, les chiffres après la virgule ont pour poids $10^{-1} = \frac{1}{10}$, $10^{-2} = \frac{1}{100}$, etc. : $0{,}375 = \frac{3}{10} + \frac{7}{100} + \frac{5}{1000}$.

En base 2, on prolonge de la même façon les poids vers la droite de la virgule, en divisant chaque fois par 2 :

poids	2^1	2^0	,	2^{-1}	2^{-2}	2^{-3}
valeur	2	1	,	$\frac{1}{2} = 0{,}5$	$\frac{1}{4} = 0{,}25$	$\frac{1}{8} = 0{,}125$

Par exemple, $(0{,}101)_2 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} = 0{,}5 + 0{,}125 = 0{,}625$.

Q2. Calculer la valeur décimale de $(0{,}11)_2$, de $(0{,}01)_2$ et de $(10{,}1)_2$.

§2. Écrire un nombre à virgule en base 2

Pour la partie entière, on utilise la méthode des divisions successives (Doc. 4,1). Pour la partie après la virgule, on fait l'inverse : on multiplie par 2. Chaque multiplication fait passer un chiffre binaire devant la virgule.

Exemple résolu : écrire 0,625 en base 2

On multiplie par 2. La partie entière du résultat donne le chiffre suivant ; on continue avec la partie après la virgule :

calcul	**chiffre obtenu**	**on continue avec**
$0{,}625 \times 2 = 1{,}25$	**1**	$0{,}25$
$0{,}25 \times 2 = 0{,}5$	**0**	$0{,}5$
$0{,}5 \times 2 = 1$	**1**	0 : on s'arrête

Les chiffres se lisent cette fois **de haut en bas** : $0{,}625 = (0{,}101)_2$. On retrouve le résultat de la partie 1.

Q3. Écrire 0,25 puis 0,75 en base 2 par la méthode des multiplications successives.

Q4. Appliquer la méthode à 0,1. Effectuer au moins six multiplications. Que remarque-t-on ?

Coup de pouce

Après quelques étapes, on retombe sur un nombre déjà rencontré. Que va-t-il se passer ensuite ?

Q5. En base 10, on sait déjà qu'un nombre simple peut avoir une écriture infinie : $\frac{1}{3} = 0{,}333\ldots$

1. Écrire $\frac{1}{3}$ en base 2 en partant de $\frac{1}{3} \times 2 = \frac{2}{3}$. Travailler avec des fractions.
2. Proposer une explication : pourquoi 0,25 et 0,625 ont-ils une écriture finie en base 2, mais pas 0,1 ?

§3. Ce que l'ordinateur range vraiment

Un flottant Python est rangé sur 64 bits. Le nombre est écrit en base 2 sous une forme qui rappelle l'écriture scientifique : des chiffres significatifs, appelés **mantisse**, multipliés par une puissance de 2. Le nombre de chiffres de la mantisse est limité. Une écriture binaire infinie, comme celle de 0,1, doit donc être **coupée** : l'ordinateur range le nombre le plus proche de 0,1 qu'il sait écrire, mais pas 0,1 lui-même.

Q6. Python arrondit l'affichage des flottants pour qu'il soit lisible. On peut demander davantage de chiffres avec une f-string (Doc. 2,2). Exécuter :

```python
>>> f"{0.1:.20f}"
>>> f"{0.25:.20f}"
>>> f"{0.1 + 0.2:.20f}"
>>> f"{0.3:.20f}"
```

Expliquer à l'aide de ces résultats pourquoi `0.1 + 0.2 == 0.3` vaut `False`.

Q7. Prévoir, puis vérifier : `0.5 + 0.25 == 0.75`. Le résultat contredit-il la règle « il ne faut pas tester l'égalité de deux flottants » ?

Ces petites erreurs peuvent avoir des conséquences graves. Le 25 février 1991, pendant la guerre du Golfe, une batterie de missiles Patriot n'a pas intercepté un missile Scud, qui a tué 28 soldats à Dhahran, en Arabie saoudite. L'horloge du système comptait le temps en dixièmes de seconde, et la valeur 0,1 était rangée sur 24 bits, donc tronquée : à chaque dixième de seconde, l'horloge prenait un retard d'environ 0,000 000 095 s.

Q8. La batterie fonctionnait sans interruption depuis 100 heures. Combien de dixièmes de seconde l'horloge avait-elle comptés ? Quel était alors son retard ? Un Scud vole à environ 1 676 m/s : quelle distance parcourt-il pendant ce retard ?

§4. Programmer avec des flottants

Q9. On considère le programme suivant :

```python
x = 0
while x != 1:
    x = x + 0.1
```

1. Combien d'itérations l'auteur de ce programme pensait-il faire ?
2. Expliquer pourquoi la boucle ne s'arrête jamais. Ne pas l'exécuter telle quelle ; pour le vérifier, exécuter plutôt :

```python
x = 0
for i in range(10):
    x = x + 0.1
print(x, x == 1)
```

Pour comparer deux flottants, on ne teste pas leur égalité. On vérifie que leur écart est très petit, par exemple inférieur à 10^{-9}. La fonction `abs` donne la valeur absolue d'un nombre, et 10^{-9} s'écrit `1e-9` en Python.

Q10. Écrire la fonction `sont_proches` dont la spécification est :

```python
def sont_proches(a: float, b: float) -> bool:
    """
    Retourne True si l'écart entre a et b est strictement
    inférieur à 1e-9, False sinon.

    >>> sont_proches(0.1 + 0.2, 0.3)
    True
    >>> sont_proches(0.3, 0.31)
    False
    """
```

Q11. Réécrire la boucle de la question précédente pour qu'elle effectue exactement 10 itérations et que x prenne les valeurs 0,1 ; 0,2 ; … ; 1 aussi précisément que possible. Justifier le choix.

> **Coup de pouce**
>
> Les entiers, eux, sont toujours exacts en Python. Quelle variable pourrait compter les itérations sans erreur ?

§5. Bilan

> **À l'issue de cette activité, je sais…**
>
> - calculer la valeur décimale d'un nombre à virgule écrit en base 2 ;
> - écrire un nombre comme 0,25, 0,1 ou $\frac{1}{3}$ en base 2 par la méthode des multiplications successives ;
> - expliquer pourquoi certains nombres, comme 0,1, ont une écriture binaire infinie et ne sont rangés en machine que de façon approchée ;
> - expliquer pourquoi `0.1 + 0.2 == 0.3` vaut `False` ;
> - éviter de tester l'égalité de deux flottants, et comparer leur écart à une petite valeur ;
> - préférer un compteur entier à l'accumulation de flottants pour contrôler une boucle.

Le Doc. 4,6 propose des exercices qui mêlent toutes les représentations étudiées dans ce chapitre.