

# Représentation approximative des nombres réels

Nous avons vu que le langage Python permet de calculer avec des nombres décimaux particuliers appelés *nombres flottants*.

Nous allons voir que ces nombres ont un encodage très compact, sur 32 ou 64 bits, qui permet de représenter des nombres très grands, bien au-delà de ce qu'il est possible de représenter avec un codage des entiers sur le même nombre de *bits*, mais également de très petits nombres.

## Introduction : L'approximation à travers un exemple concret

### 1 L'exemple de l'addition 0.1+0.2 :

- Calculer **0.1+0.2** avec une console Python.
- Pourtant, en réalisant ce calcul sur ordinateur, vous remarquerez que le résultat est quelque chose comme 0.30000000000000004 au lieu de 0.3.

### 2 Que se passe-t-il exactement ? :

- Ce phénomène est dû à la manière dont les ordinateurs (et les calculatrices) représentent les nombres réels. Les ordinateurs ne peuvent pas stocker tous les nombres réels exactement avec une précision infinie.
- Les nombres décimaux comme 0.1 et 0.2 n'ont pas de représentation exacte en binaire dans le système de base utilisé par les ordinateurs. Ce sont des approximations qui introduisent des erreurs.

### 3 Introduction du concept de représentation binaire :

- Les ordinateurs utilisent le système binaire (base 2) pour stocker les nombres, et certains nombres décimaux comme 0.1 ne peuvent pas être exprimés de manière exacte en binaire.
- Par exemple, 0.1 en base 10 est approximativement égal à 0.00011001100110011... en binaire, avec une infinité de chiffres après la virgule. De cette manière, les ordinateurs doivent couper cette représentation après un certain nombre de chiffres, ce qui crée une petite erreur.

### 4 La notion d'erreur d'approximation :

- Introduis le concept de **précision** : dans un contexte informatique, on ne peut travailler qu'avec un nombre limité de chiffres après la virgule (souvent 15 à 16 chiffres en double précision).

- En conséquence, les résultats d'opérations comme  $0.1+0.2$  ne sont pas toujours exacts, et c'est cette erreur qui donne un résultat légèrement différent de  $0.3$ .

## Conversion de 0.1 en base 10 en binaire

Lorsque nous convertissons un nombre décimal en binaire, nous procédons à un processus similaire à la division par 2 (pour les entiers), mais pour la partie fractionnaire, nous multiplions successivement par 2 et extrayons la partie entière à chaque étape. Voici comment ça fonctionne pour 0.1 en base 10 :

### 1. Multiplier 0.1 par 2 :

$$0.1 \times 2 = 0.2 \Rightarrow \text{partie entière} = 0, \text{ reste} = 0.2$$

On garde la partie entière 0, et on répète l'opération avec le reste 0.2.

### 2. Multiplier 0.2 par 2 :

$$0.2 \times 2 = 0.4 \Rightarrow \text{partie entière} = 0, \text{ reste} = 0.4$$

On garde la partie entière 0, et on répète l'opération avec 0.4.

### 3. Multiplier 0.4 par 2 :

$$0.4 \times 2 = 0.8 \Rightarrow \text{partie entière} = 0, \text{ reste} = 0.8$$

### 4. Multiplier 0.8 par 2 :

$$0.8 \times 2 = 1.6 \Rightarrow \text{partie entière} = 1, \text{ reste} = 0.6$$

On garde la partie entière 1, et on répète avec 0.6.

### 5. Multiplier 0.6 par 2 :

$$0.6 \times 2 = 1.2 \Rightarrow \text{partie entière} = 1, \text{ reste} = 0.2$$

À ce point, on observe que nous revenons à 0.2, donc le processus commence à se répéter.

## Résultat : 0.1 en binaire

En combinant les parties entières extraites à chaque étape, on obtient la représentation binaire de 0.1 comme une fraction périodique :

$$0.1_{10} = 0.00011001100110011 \dots_2$$

Cette séquence de chiffres binaires se répète indéfiniment avec "1100". En d'autres termes, 0.1 en base 10 ne peut pas être exactement représenté en binaire, car la fraction binaire qui la représente est infinie et périodique.

## Ecriture scientifique

L'encodage des nombres flottants est inspiré de l'écriture scientifique des nombres décimaux qui se compose d'un signe (+ ou -), d'un nombre décimal  $m$ , appelé *mantisse*, compris dans l'intervalle  $[1, 10[$  (1 inclus et 10 exclu) et d'un entier relatif  $n$  appelé *exposant*.

Par exemple, avec cette notation :

2156 s'écrit  $+2,156 \times 10^3$

-398 879,62 s'écrit  $-3,988\,796\,2 \times 10^5$

0,000 142 s'écrit  $+1,42 \times 10^{-4}$

## Format IEEE 754

La représentation des nombres *flottants* et les opérations arithmétiques qui les accompagnent ont été définies dans la norme internationale IEEE 754. C'est la norme la plus couramment utilisée dans les ordinateurs.

Selon la précision ou l'intervalle de représentation souhaité, la norme définit un format de données sur 32 bits, appelé *simple précision* et un autre sur 64 bits, appelé *double précision*.

Dans les deux cas, la représentation d'un nombre flottant est similaire à l'écriture scientifique d'un nombre décimal, à savoir une décomposition en 3 parties : un signe  $s$ , une mantisse  $m$  et un exposant  $n$ . De manière générale, un nombre flottant a la forme suivante :

$$(-1)^s m \times 2^{(n-d)}$$

Le format de 32 bits est composé de trois parties :

### 1. Bit de signe (1 bit) :

- 0 : le nombre est positif.
- 1 : le nombre est négatif.

### 2. Exposant (8 bits) :

- L'exposant est stocké sous forme de **valeur décalée** (ou "biased") en ajoutant un biais de **127** au véritable exposant.
- Cela permet de représenter à la fois des exposants positifs et négatifs.
- La valeur de l'exposant est donc la valeur réelle de l'exposant + 127.

### 3. Mantisse (ou fraction, 23 bits) :

- La mantisse représente la partie significative du nombre.

- En **normale**, la mantisse est toujours un nombre compris entre 1 et 2, donc la première valeur binaire avant la virgule est toujours **1** (et n'est pas stockée explicitement, ce qu'on appelle le "1 implicite").
- Les 23 bits suivants représentent la fraction binaire après la virgule.

### **Étapes de l'encodage d'un nombre décimal en binaire IEEE 754 (32 bits)**

#### **1. Déterminer le signe :**

- Si le nombre est positif, le bit de signe est **0**.
- Si le nombre est négatif, le bit de signe est **1**.

#### **2. Convertir la partie entière en binaire :**

- Effectuer la conversion de la partie entière du nombre en binaire (comme pour tout nombre entier).

#### **3. Convertir la partie décimale en binaire :**

- Multiplier la partie décimale par 2.
- La partie entière de ce produit devient le premier bit après la virgule.
- Répéter ce processus avec la nouvelle partie décimale jusqu'à obtenir un nombre suffisamment précis ou jusqu'à ce que l'on atteigne les 23 bits de la mantisse.

#### **4. Normaliser le nombre en binaire :**

- Convertir le nombre en une forme normalisée (sous la forme  $1.xxxxx \times 2^{\text{exposant}}$ ).
- Exemple : 6.25 devient  $1.25 \times 2^2$ .

#### **5. Calculer l'exposant :**

- L'exposant réel est l'exposant de la forme normalisée.
- Ajouter le biais de 127 à cet exposant pour obtenir l'exposant stocké dans le format IEEE 754.
- Exemple : si l'exposant réel est 2, l'exposant stocké sera  $2 + 127 = 129$ .

#### **6. Construire la mantisse :**

- Exclure le "1" implicite devant la virgule et utiliser seulement les 23 premiers bits après la virgule de la partie fractionnaire du nombre normalisé.

#### **7. Assembler le tout :**

- Combiner le bit de signe, les 8 bits de l'exposant et les 23 bits de la mantisse pour obtenir la représentation binaire finale en 32 bits.

### **Exemple détaillé : Convertir 6.25 en IEEE 754 32 bits**

- 1. Signe :** Le nombre est positif, donc le bit de signe est 0.
- 2. Convertir la partie entière (6) en binaire :**
  - 6 en binaire est 110.
- 3. Convertir la partie décimale (0.25) en binaire :**
  - Multiplier 0.25 par 2 :  $0.25 * 2 = 0.5$  (partie entière 0).
  - Multiplier 0.5 par 2 :  $0.5 * 2 = 1.0$  (partie entière 1).
  - La partie décimale 0.25 en binaire est 0.01.
- 4. Forme normalisée :**
  - Combinez la partie entière et décimale en binaire : 6.25 devient 110.01 en binaire.
  - En forme normalisée, cela devient  $1.1001 * 2^2$ .
- 5. Calcul de l'exposant :**
  - L'exposant réel est 2.
  - Ajouter le biais de 127 :  $2 + 127 = 129$ .
  - L'exposant en binaire de 129 est 10000001.
- 6. Mantisse :**
  - La partie fractionnaire de la forme normalisée est 1001.
  - Remplir avec des zéros pour obtenir 23 bits : 10010000000000000000000.
- 7. Assembler le tout :**
  - Bit de signe : 0
  - Exposant : 10000001
  - Mantisse : 10010000000000000000000

Le nombre binaire final en IEEE 754 32 bits est :

0 10000001 10010000000000000000000

## Résumé de la structure IEEE 754 32 bits

Signe (1 bit)	Exposant (8 bits)	Mantisse (23 bits)
0	10000001	10010000000000000000000

### Exemple inverse

Par exemple, le mot de 32 *bits* suivant

$\overbrace{1}^{\text{signe}} \quad \overbrace{10000110}^{\text{exposant}} \quad \overbrace{10101101100000000000000}^{\text{fraction}}$

représente le nombre décimal calculé ainsi :

$$\begin{aligned} \text{signe} &= -1^1 \\ &= -1 \end{aligned}$$

$$\begin{aligned} \text{exposant} &= (2^7 + 2^2 + 2^1) - 127 \\ &= (128 + 4 + 2) - 127 \\ &= 134 - 127 \\ &= 7 \end{aligned}$$

$$\begin{aligned} \text{mantisse} &= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-8} + 2^{-9} \\ &= 1,677734375 \end{aligned}$$

Soit, au final, le nombre décimal suivant :

$$-1,677734375 \times 2^7 = -214,75$$

## En résumé :

### Format IEEE 754 - Simple Précision (32 bits)

Le format IEEE 754 pour les nombres à virgule flottante en **simple précision** utilise **32 bits** répartis en trois parties :

- **1 bit de signe** : détermine si le nombre est positif ou négatif.
- **8 bits pour l'exposant** : utilisé pour représenter la puissance de 2 à laquelle le nombre est élevé.
- **23 bits pour la mantisse (ou fraction)** : la partie significative du nombre.

Le nombre est représenté par l'expression suivante :

$$(-1)^s \times 1.M \times 2^{E-127}$$

- $s$  : bit de signe (0 pour positif, 1 pour négatif)
- $M$  : mantisse (la partie après la virgule)
- $E$  : exposant, qui est stocké avec un **décalage de 127** (c'est-à-dire qu'on y ajoute 127 pour éviter des valeurs négatives dans le format binaire).

## Questions

### Exercice 1

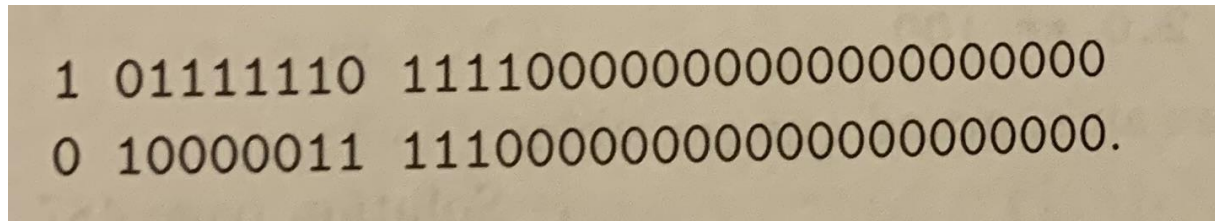
- Convertir 5.75 en IEEE 754
- Convertir -9,42 en IEEE 754

### Exercice 2

Donner la représentation flottante en simple précision de 128 et -32,75.

### Exercice 3

Donner la valeur décimale des nombres flottants suivants codés en simple précision :



1 01111110 11110000000000000000000000000000  
0 10000011 11100000000000000000000000000000.