Encodage des réels

I – Ensemble des réels et encodage dans Python

Les **entiers naturels** sont les entiers positifs. Ils appartiennent à l'ensemble ℕ. Ex : 5

Les **entiers relatifs** sont les entiers positifs ou négatifs. Ils appartiennent à l'ensemble ℤ. Ex : +5.

Les **décimaux** caractérisent l'ensemble des nombres qui ne possèdent qu'un nombre limité de chiffres après la virgule. Ils peuvent tous s'exprimer comme le quotient d'un entier par une puissance de 10. Ils appartiennent à l'ensemble 𝔻. Ex : 5.0

Les **rationnels** caractérisent l'ensemble des nombres qu'on peut écrire comme un quotient de deux nombres entiers relatifs. Ils appartiennent à l'ensemble noté ℚ. Ex : 5/1 ou 5/2 ou 1/3.

Les **irrationnels** sont les nombres qui ne sont pas rationnels : on ne peut pas les exprimer comme une quotient de deux entiers. On peut y placer Pi, racine de 2 ou racine de 17 par exemple.

Les **réels** caractérisent donc l'ensemble des nombres : les nombres rationnels et les nombres irrationnels. Ils appartiennent à l'ensemble ℝ.

Python encode les nombres soit comme des entiers (type **`int`**), soit comme des réels (type **`float`**).

Il faut être vigilant sur les calculs avec les floats car les ordinateurs ne parviennent pas toujours à stocker exactement la valeur mais une approximation de la valeur.

C'est compréhensible pour racine de 2, moins pour 0,1… Bizarrement, les calculs sont parfois exacts !

```
>>> import math
>>> x = math.sqrt(2) # Racine de 2
>>> x
1.4142135623730951
>>> x * x   # On devrait obtenir 2 !
2.0000000000000004
```

```
>>> 0.1
0.1
>>> 3 * 0.1  # 0.3 ?
0.30000000000000004
```

```
>>> 1/8
0.125
>>> 3 * 0.125
0.375
```

II – Représentation d'un réel en base 10 : écriture scientifique

N = 0.1 est un décimal car N = 1/10 = 1 x 10^{-1}
N = 0.15 est un décimal car N = 1 x 10^{-1} + 5 x 10^{-2} = 15 x 10^{-2}

N = 1/3 n'est pas un décimal : on ne peut pas l'écrire EXACTEMENT comme une puissance de 10.
Par contre, on peut représenter APPROXIMATIVEMENT 1/3 sous la forme N = 0,33333333333333333….

III – Représentation d'un réel en base 2 : écriture scientifique

1) Puissance positive et négative de 2

2^0 = _____

2^1 = _____ 2^2 = _____ 2^3 = _____ 2^4 = _____ 2^5 = _____ 2^6 = _____

2^{-1} = _____ 2^{-2} = _____ 2^{-3} = _____ 2^{-4} = _____ 2^{-5} = _____ 2^{-6} = _____

2) Exemple simple de représentation en base 2

Exemple : Trouver la valeur N encodée en base 2 par **1010,1011** $_2$ Attention à la virgule !

Bit encode	8	4	2	1	0,5	0,25	0,125	0.0625
	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
BITS EN MEMOIRE								

On obtient en base 10 : N =

Exercice 01 : Trouver la valeur N encodée en base 2 par **101,011** $_2$

Exercice 02 : Trouver la valeur N encodée en base 2 par **1111,1111** $_2$

Exercice 03 : Calculer la valeur du réel **0,750** $_{10}$ exprimé en base 2. Vous utiliserez la technique de l'activation progressive des bits les plus forts.

3) Algorithme pour trouver la représentation d'un nombre réel positif inférieur à 1

Vous pouvez appliquer la méthode suivante plutôt que celle de l'activation des bits de poids forts.

- **Premier bit avant la virgule ← 0**
- **nombre ← nombre voulu**
- **TANT QUE nombre > 0**
    - **nombre ← nombre * 2**
    - **bit suivant ← partie entière de nombre (0 ou 1)**
    - **SI nombre >= 1**
        - **nombre ← nombre – 1**

Implanté en Python dans une fonction, on peut obtenir ceci

```
1   def decomposition(nombre):
2       '''Décompose un réel dans [0;1[ en bits
3
4       ::param nombre(float) :: nombre est un réel dans l'intervalle [0;1[
5       ::return (str) :: renvoie un string contenant la décomposition
6
7       '''
8       bits = "0,"
9       while nombre > 0:
10          nombre = nombre * 2
11          bits = bits + str(nombre)[0] # On rajoute l'unité au string
12          if nombre >= 1 :
13              nombre = nombre - 1
14      return bits
15
16  if __name__ == '__main__' :
17      print(decomposition(0.5))
18      print(decomposition(0.25))
```

Exercice 04 : Utiliser cet algorithme pour trouver les bits de la décomposition de 0.375.

Exercice 05 : Utiliser cet algorithme pour trouver les 10 premiers bits de la décomposition de 0.1. Pourquoi les calculs sur 0,1 sont-ils approximatifs) votre avis par rapport aux calculs utilisant 0,375 ?

Exercice 06 : Pourquoi le programme donne-t-il parfois des résultats approximatifs alors qu'il n'est que la transcription d'un algorithme exact ? Utiliser le programme pour trouver la décomposition de 1/3.

On retiendra donc qu'il ne faut JAMAIS réaliser de tests d'EGALITE sur des nombres stockés sous forme de FLOAT car on ne stocke pas toujours la valeur exacte mais une valeur approximative de la valeur fournie initialement.

IV - Notation scientifique en puissance de 2

La notation scientifique permet d'écrire un nombre sous forme

- du signe
- d'un nombre m (nommé **mantisse**) appartenant à **[1; 2[** qu'on multiplie par
- une puissance de 2 où E désigne l'**exposant**.

On pourra donc noter : $\mathbf{N = signe \times m \times 2^E}$

L'exposant correspond à la puissance de 2 juste inférieure ou égale au nombre lui-même. Pour les cas un peu compliqués, on peut le trouver facilement une fois obtenue la décomposition du nombre en puissance de 2.

Exemple 1 Prenons **N = 5,0** Nous savons que $N = 5,0_{10} = \underline{1}01,0_2$

On constate que le premier 1 est décalé de 2 positions par rapport à l'unité : il encode 2^2.

On peut donc écrire : $N = 5,0_{10} = 1,01_2 \times 2^2$

Exemple 2 Prenons **N = 0,5** Nous savons que $N = 0,5_{10} = 0,\underline{1}_2$

On constate que le premier 1 est décalé de 1 positions par rapport à l'unité : il encode 2^{-1}.

On peut donc écrire : $N = 5,0_{10} = 1,00_2 \times 2^{-1}$

Exercice 10 : Exprimer 0,125 sous cette forme.

Exercice 11 : Exprimer 0,1 sous cette forme. Rappel : $\mathbf{N = 0.1}_{10} = 0,000\ \underline{1}100\ 1100\ 1100_2$

Exercice 12 : Exprimer 1/3 sous cette forme. Rappel : $N = 1/3_{10} = 0,0\underline{1}\ 01\ 01\ 01\ 01_2$

V – Encodage FLOAT

Il existe deux encodages principaux des FLOATS. **A connaître uniquement en tant que culture générale.**

L'encodage **SIMPLE PRECISION** travaille avec 32 bits (4 octets).
→ 1 BIT pour le signe (0 pour positif, 1 pour négatif)
→ 8 BITS formant un entier naturel D pour gérer l'exposant E. **E = D – 127.**
→ 23 BITS pour les bits derrière la virgule de la mantisse (on ne stocke pas le 1 car on a toujours 1, …)

L'encodage **DOUBLE PRECISION** travaille avec 64 bits (8 octets).
→ 1 BIT pour le signe (0 pour positif, 1 pour négatif)
→ 11 BITS pour gérer l'exposant
→ 52 BITS pour les bits derrière la virgule de la mantisse (on ne stocke pas le 1 car on a toujours 1, …)

Exercice 13 : Trouver les bits du signe, de l'exposant et de la mantisse sur le nombre suivant. Trouver la valeur de D, en déduire la valeur de l'exposant. Trouver ensuite la valeur de la mantisse m. En déduire le nombre encodé.

`0011 1110 0000 0000 0000 0000 0000 0000`

`0 **011 1110 0** 000 0000 0000 0000 0000 0000`

Exercice 16 : Idem avec les 32 bits suivants :

`1011 1101 1100 1100 1100 1100 1100 1100`

`1 **011 1101 1** 100 1100 1100 1100 1100 1100`