Nous avons vu comment représenter des nombres entiers naturels et relatif (en base 2,10 et 16). Nous allons maintenant apprendre à représenter les nombres à virgule flottante qu'ils soient positifs ou négatifs : **les flottants**.

# Représentation des flottants (positifs ou négatifs)

## Expérience

Ouvrir une console python et tester les codes suivants :

* *0.2+0.3==0.5* réponse
* *0.1+0.2==0.3* réponse

Remarque :

* *0.1+0.2* réponse

Nous allons essayer de comprendre pourquoi la console Python réagit ainsi.

## Décomposition d'un nombre flottant en base décimale

Voici un nombre flottant : 21.3786, décomposons ce nombre en base 10 :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 101 | 100 |  | 10-1 | 10-2 | 10-3 | 10-4 |
|  |  | . | 3 |  |  |  |

21.378610 = 2 x 101 + 1 x 100 + 3 x 10-1 +

En notation décimale, les chiffres à droite de la virgule représentent les dixièmes, les centièmes, les millièmes, etc

## Conversion d'un nombre flottant en base binaire vers le décimal

En utilisant la même méthode, on peut écrire le nombre flottant binaire 101.1101

Les chiffres à droite de la virgule représentent les demis, les quarts, les huitièmes, les seizièmes, etc.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 22 | 21 | 20 |  | 2-1 | 2-2 | 2-3 | 2-4 |
|  |  |  | . |  |  |  |  |

101.11012 = 1 x 22 + 0 x 21 + 1 x 20  +

101.11012 = 1 x 22 + 1 x 20  +

**Conclusion : À droite de la virgule,**

**Exercice :**

Trouver la valeur décimale des nombres suivants :

10111.101012 =

11101.0111012 =

## Conversion d'un nombre flottant en base décimal vers le binaire

Pour convertir un nombre décimale 7.8125 en binaire, on commence par convertir la partie entière (à gauche de la virgule) : 7 = (111)2.

Pour la partie à droite, on fait des multiplications successives par 2 sans reporter la partie entière :

0,8125 × 2 =

0,625 × 2 =

0,25 × 2 =

0,5 × 2 =

On trouve donc : 7.812510 =

**Exercice :**

Trouver la valeur binaire des nombres décimaux suivants :

65.340 =

0.379352 =

**Conclusion :** Il existe des nombres dont l’écriture binaire sera infinie (et périodique). Par exemple, 1,2 = (1,001100110011 … )2 , le cycle « 0011 » se répétant à l’infini. Comme on ne peut pas représenter en machine un mot infini, il ne sera pas possible de représenter de manière exacte certains nombres réels (beaucoup d’entre eux !).

## Représentation scientifique des nombres flottants

Dans un ordinateur, les nombres à virgule (réels) sont codés en virgule flottante. On parle de nombres flottants (le type float de Python).

La représentation binaire en machine d’un nombre flottant s’inspire de l’écriture scientifique des nombres décimaux dont voici quelques rappels.

Écriture scientifique L’écriture scientifique permet d’uniformiser la façon d’écrire des nombres décimaux.

Par exemple : 3542 s’écrit +3,542 × 103

−0,0724753 s’écrit −7,24753 × 10−2

Dans cette écriture, on distingue :

* + Un signe (+ ou −) ;
  + Un nombre décimal, appelé mantisse, compris dans l’intervalle [1; 10[ (1 inclus et 10 exclu). Dans les deux exemples, il s’agit de 3,542 et 7,24753;
  + Un entier relatif 𝑛, appelé exposant. Dans les deux exemples, il s’agit de 3 et −2. Ainsi, de manière générale, l’écriture scientifique d’un nombre décimal est : **±𝑚 × 10𝑛**

**La norme IEEE 754**

La norme IEEE 754 est la plus utilisée pour représenter les nombres flottants. Ils sont représentés sur 32 bits (format appelé « simple précision » ou binary32) ou sur 64 bits (format appelé « double précision », ou binary64) sous la forme :

**𝑠. 𝑚 × 2𝑛** où :

* + 𝑠 est le signe du nombre, codé sur 1 bit (0 pour + et 1 pour −) ;
  + 𝑛 son exposant en puissance de 2, codé sur 8 bits (en format 32 bits) ou sur 11 bits (en format 64 bits) ;
  + 𝑚 sa mantisse codée sur 23 bits (en format 32 bits) ou sur 52 bits (en format 64 bits). Ainsi, en machine, un flottant est représenté en format 32 bits (simple précision) par un mot binaire de la forme

|  |  |  |
| --- | --- | --- |
| 1 bit | 8 bits | 23 bits |
| signe | exposant | mantisse |

et en format 64 bits (double précision) par un mot binaire de la forme

|  |  |  |
| --- | --- | --- |
| 1 bit | 11 bits | 52 bits |
| signe | exposant | mantisse |

**Exemple :**

Reprenons le nombre 7.8125:

On sait que 7.8125 est positif donc le bit de signe sera 0 : signe exposant mantisse 0

Rappelons que 7.8125 = (111.1101)2 . Par analogie avec l’écriture scientifique, on peut aussi écrire ce nombre binaire : 1,111101 × 22 en décalant la virgule de deux rangs vers la gauche.

En faisant cela, on a fait apparaître :

* + la mantisse : 𝑚 = 1,111101
  + l’exposant : 2

Il reste maintenant à voir comment sont codés la mantisse et l’exposant.

**Codage de la mantisse**

Pour représenter les flottants, la base choisie est la base 2 (contrairement à l’écriture scientifique qui est la base 10) donc la mantisse est dans l’intervalle [1; 2[. Il s’agit donc d’un nombre de la forme : 𝑚 = 1, 𝑥𝑥 … 𝑥𝑥

Comme cette mantisse commence toujours par le chiffre 1, il a été choisi de ne pas coder ce « 1 » mais uniquement les chiffres après la virgule.

La mantisse de ce nombre est 𝑚 = 1,111101. Comme le « 1 » à gauche de la virgule n’est pas codé, la mantisse sera codée par 11110100…0 en ajoutant autant de zéros que nécessaires pour arriver à 23 bits (simple précision) ou 52 bits (double précision).

**Codage de l’exposant**

L’exposant est codé sur 8 bits ou 11 bits selon le format utilisé.

Sur 8 bits on peut coder 256 valeurs : les entiers compris entre −127 et 128.

Sur 11 bits on peut coder 2048 valeurs : les entiers compris entre −1023 et 1024.

L’exposant est un entier relatif mais la norme IEEE 754 n’utilise pas l’encodage par complément à 2 des entiers relatifs. Elle prévoit un décalage qui dépend de l’encodage utilisé :

• Dans le format simple précision (32 bits), le décalage est 127, c’est-à-dire qu’il faut ajouter 127 à l’exposant.

• Dans le format double précision (64 bits), le décalage est de 1023.

L’objectif est d’obtenir un nombre positif pour coder l’exposant. En effet, dans le format simple précision, en procédant à ce décalage de 127, on obtient des valeurs positives :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Exposants signés | -127 | -126 | … | 0 | … | 127 | 128 |
| Entier n coder |  |  |  |  |  |  |  |

Dans notre exemple, il faut coder l'exposant +2, donc 2 +127 =

129 =

Dans le format 32 bits :

|  |  |  |
| --- | --- | --- |
| Signe | Exposant | Mantisse |
| 1 bit | 8 bits | 23 bits |
| 0 | 1000 0001 | 1111 0100…0 |

Donc la représentation du nombre 7.8125 au format 32 bits est :

**0 10000001 11110100000000000000000**

Au format 64 bits :

**0 10000000001 1111010000000000000000000000000000000000000000000000**

## Représentation approximative

**Codage approché de certains réels**

Tous les nombres réels dont l’écriture en base est infinie ne peuvent être représentés de manière exacte en machine. Par exemple, le nombre décimal 1,2 a une écriture binaire infinie : 1,2 = (1,001100110011 … )2 = +1.001100110011 … × 20

La mantisse de ce nombre est 𝑚 = 1,001100110011 … donc elle est infinie. Or, il n’y a que 23 ou 52 bits réservés pour coder la mantisse. L’ordinateur doit donc tronquer la mantisse à 23 ou 52 bits. Cela signifie qu’en format simple précision, la mantisse du nombre 1.2 est codée par le mot binaire de 23 bits 001100110011001100110011001 ~~100110011~~ … sur 23 bits, les autres bits ne pouvant pas être codés.

Seule une valeur tronquée de la mantisse peut être codée et donc la représentation en machine du nombre réel 1.2 n’est qu’une approximation du réel 1.2. Autrement dit, **le nombre flottant 1.2 n’est qu’une valeur approchée du nombre réel 𝟏.𝟐**

Cet exemple n’en est qu’un parmi tant d’autres : il y a une infinité de nombres réels qu’il est impossible de représenter de manière exacte en machine.

**Impossibilité de coder tous les nombres réels**

Voici l’écriture binaire en format double précision de deux flottants.

Nombre flottant Représentation format double précision (mantisse sur 52 bits)

1.5 +1.1000000000000000000000000000000000000000000000000000 × 20 1.5000000000000002 +1.1000000000000000000000000000000000000000000000000001 × 20

On comprend aisément qu’il n’y a pas de flottant entre 1.5 et 1.5000000000000002.

Le flottant 1.5000000000000001 est donc représenté comme 1.5.

La précision possible avec une mantisse sur 52 bits se situe au niveau dernier bit qui vaut 2−52 soit environ 2 × 10−16 . L’écart entre 1.5 et 1.5000000000000002 étant égal à 2 × 10−16, on ne peut pas trouver un flottant compris entre les deux.

***Conclusion :*** Les nombres flottants sont une représentation approximative des nombres réels dans un ordinateur. En particulier, il n’est pas possible de représenter de manière exacte en machine tous les nombres réels. La manipulation de nombres réels par un langage informatique est donc à prendre avec précaution car elle peut engendrer des résultats surprenants, en particulier **il ne faut jamais tester l’égalité entre deux flottants.**