



Qu'est-ce qu'un nombre flottant ?

Les nombres réels sont codés selon la norme IEEE754 en informatique. On obtient alors un nombre à virgule flottante. Ca vous fait une belle jambe ? Si vous ne souhaitez pas en comprendre le principe, sachez simplement que ce codage engendre des approximations, même sur des décimaux "simples" comme 0,1 par exemple, que ces approximations sont sources de problèmes lors de la comparaison entre nombres à virgules flottantes et passez de suite à [la comparaison de nombre à virgules flottantes](#).

En base 10

Si vous êtes encore là, vous voulez en savoir plus. Commençons par un nombre décimal en base 10. par exemple 223,25.

- Sa partie entière est 223. Pour la décomposer en unités, dizaines et centaines, on effectue des divisions successives par 10 jusqu'à obtenir un quotient nul. Les restes nous donnent les unités, dizaines et centaines.

223		10	
3		22	10
		2	2 10
		2	0

Ainsi $223 = 2 * 10^2 + 2 * 10^1 + 3 * 10^0$ soit 223 en base 10.
(noter qu'on prend les restes de droite à gauche pour obtenir 223)

- Pour sa partie décimale, on multiplie par 10 jusqu'à ce qu'il n'y en ait plus et on conserve la partie entière obtenue :

$0,25 * 10 = 2,5 \rightarrow$ on obtient 2 dixièmes, soit $2 * 10^{-1}$.

$0,5 * 10 = 5,0 \rightarrow$ on obtient 5 centièmes, soit $5 * 10^{-2}$ et la partie décimale est nulle donc c'est terminé.

- Reste à décaler la virgule, en notation scientifique $223,25 = 2,2325 * 10^2$



Stage Python

Python et les nombres flottants



En base 2

C'est exactement le même principe avec 2 à la place de 10.

- occupons nous de la partie entière

$$\begin{array}{r} 223 \mid \underline{2} \\ 1 \mid 111 \mid \underline{2} \\ 1 \mid 55 \mid \underline{2} \\ 1 \mid 27 \mid \underline{2} \\ 1 \mid 13 \mid \underline{2} \\ 1 \mid 6 \mid \underline{2} \\ 0 \mid 3 \mid \underline{2} \\ 1 \mid 1 \mid \underline{2} \\ 1 \mid 0 \end{array}$$

Ainsi, $223 = 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
(notez qu'on prend les restes de droite à gauche)
soit 11011111 en base 2

- Voyons maintenant la partie décimale :

$0,25 \cdot 2 = 0,5 \rightarrow$ partie entière 0

$0,5 \cdot 2 = 1,0 \rightarrow$ partie entière 1 et la partie décimale est nulle donc c'est terminée.

253,25 s'écrit donc 11011111,01 en base 2

- reste à décaler la virgule

$11011111,01 = 1,101111101 \cdot 2^7$ (multiplier par 2 en base 2 revient à décaler la virgule)

Ici aucune approximation ! Jusque là, tout va bien...



Stage Python

Python et les nombres flottants



Le soucis est qu'il est possible que multiplier par 2 ne permette jamais d'annuler la partie décimale.

Cela arrive même avec des décimaux simples comme, par exemple 0,1.

$0,1 * 2 = 0,2 \rightarrow$ partie entière **0**

$0,2 * 2 = 0,4 \rightarrow$ partie entière **0**

$0,4 * 2 = 0,8 \rightarrow$ partie entière **0**

$0,8 * 2 = 1,6 \rightarrow$ partie entière **1**

$0,6 * 2 = 1,2 \rightarrow$ partie entière **1**

$0,2 * 2 = 0,4 \rightarrow$ partie entière **0**

$0,4 * 2 = 0,8 \rightarrow$ partie entière **0**



0.1 en base 10 correspond à 0.000110011001100110011.... en base 2

L'ordinateur ne sait donc pas écrire 0.1 sans approximation !

Pourquoi utiliser ce codage alors ? Parce qu'il est très très rapide et que l'erreur potentielle sur un nombre est faible. Notez que ce soucis n'est pas lié à Python mais au codage des flottants par l'ordinateur.

Comparaison de nombres à virgule flottante

- Utilisons tout d'abord la console.
 - Tapez `0,1+0,1+0,1` ... Python ne renvoie pas 0,3 mais
 - si on demande "*est-que 0,1+0,1+0,1=0,3 ?*" en Python , c'est à dire si on tape `0.1+0.1+0.1==0.3` dans la console, il y a donc une surprise

C'est tout sauf un détails !

Tous les algos ayant pour but de comparer deux nombres flottants (Ce triangle est-il rectangle ? Ces deux vecteurs sont-ils colinéaires ? Ces deux fonctions prennent-elles les mêmes valeurs pour x=...) risquent de ne pas donner les résultats attendus.



Stage Python

Python et les nombres flottants



Algo : comparer deux fonctions "normalement" égales partout

- Réalisons un petit algo comportant trois fonctions :
 - une fonction $f1(x)$ renvoyant la valeur de $(x-1)*(x+1)$
 - une fonction $f2(x)$ renvoyant la valeur de x^2-1
 - une fonction $test(x)$, renvoyant vrai si les deux fonctions précédentes prennent la même valeur en x et faux sinon

Vous trouverez la solution à la fin de ce TP, ne regardez pas tout de suite !

- Dans la console,
 - tapez $test(1)$. La réponse est
 - tapez $test(0.1)$. La réponse est

Il est obligatoire de passer par des approximations. Deux flottants ne seront pas strictement égaux mais approximativement égaux. Heureusement, la précision est tout de même assez grande...

- Retournez dans notre algo pour y ajouter une fonction $approx(float1, float2, precision)$ qui renvoie vrai si $|float1 - float2| < precision$ et faux sinon.
- Corrigez la fonction $test$
- refaites les tests dans la console avec différentes valeurs de la précision.

Vous aurez besoin de la bibliothèque **math** (`import math`) et de la fonction **math.fabs(x)** qui renvoie la valeur absolue de x .

Vous trouverez la solution à la fin de ce TP, ne regardez pas tout de suite !



Stage Python

Python et les nombres flottants



Les modules "decimal" et "fractions"

Python propose également un module *decimal* et un module *fractions*. La syntaxe est un peu pénible mais cela peut être utile.

- Dans la console, tapez :
 - *from decimal import Decimal*
 - *Decimal("0,1")+Decimal("0,1")+Decimal("0,1") ...*
- Dans la console, tapez :
 - *from fractions import Fraction*
 - *Fraction (1,3)+Fraction (1,4)* c'est à dire $1/3 + 1/4$...

Correction algo no1 page suivante...



Stage Python

Python et les nombres flottants



Algo no 1

```
def f1(x):  
    return (x-1)*(x+1)  
  
def f2(x):  
    return x*x-1 # on peut aussi écrire x**2-1, le ** étant l'opérateur puissance  
  
def test(x):  
    return f1(x)==f2(x)
```

Algo no2 page suivante...



Stage Python

Python et les nombres flottants



Algo no 2

```
import math

def approx(flot1,flot2,precision):
    return math.fabs(flot1-flot2)<precision

def f1(x):
    return (x-1)*(x+1)

def f2(x):
    return x*x-1 # on peut aussi écrire x**2-1, le ** étant l'opérateur puissance

def test(x):
    return approx(f1(x),f2(x),0.0001) # on peut rendre la précision plus petite
```