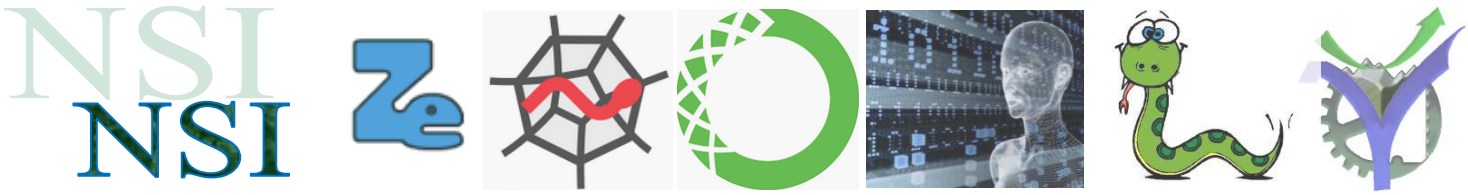
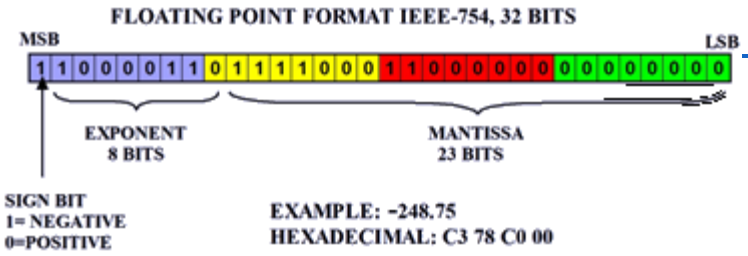


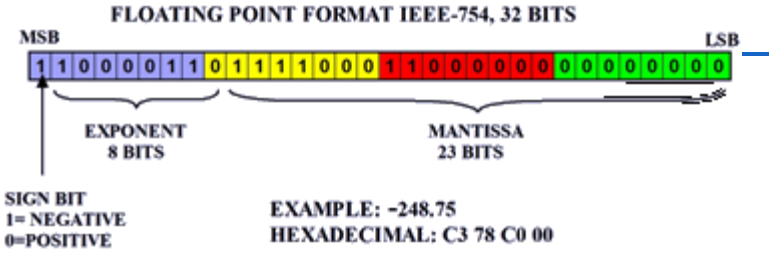
Les calculs numériques au risque des ordinateurs



Résumé :

Sommaire :

Les calculs numériques au risque des ordinateurs



Les entiers naturels N

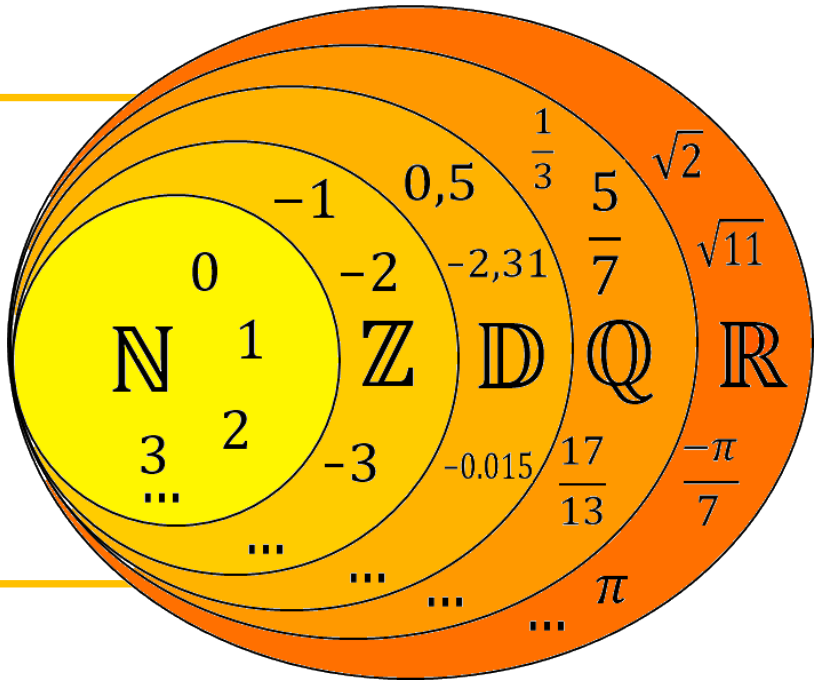
Les entiers relatifs Z

Les décimaux relatifs D

Les rationnels Q

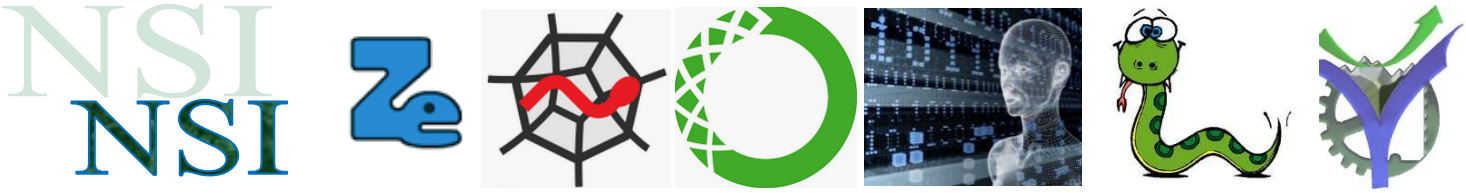
Les réels R

Les nombres flottants F



$$(-1)^{signe} \times 0.[mantisse] \times 2^{(Exposant-Decalage)}$$

Les erreurs les plus dangereuses sont idiotes



1 Le codage des nombres dans les ordinateurs[1]

1.1 Le codage des nombres entiers dans les ordinateurs

a) Le codage des nombre entiers naturels

Ce codage a déjà été étudié. En complément notons les notions suivantes pour connaitre le nombre de bits p nécessaire au codage il suffit de calculer : $p = \left\lfloor \frac{\ln n}{\ln 2} \right\rfloor$

Le nombre codé en base 2 obtenu par les divisions successives par exemple est codé :

n = 237, p = 7, n = 1110 1101

Rappel sur les notations :

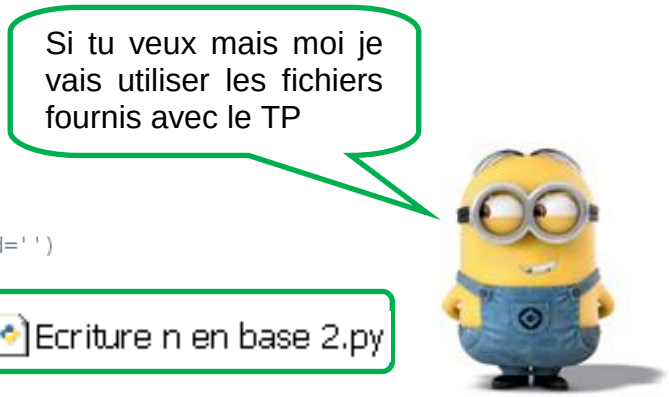
Arrondi supérieur de x $\lceil x \rceil$

Arrondi au plus proche de x $\lfloor x \rceil$

Arrondi inférieur de x $\lfloor x \rfloor$

Q1.1 : Coder et tester le script ci-dessous :

```
# -*- coding:Utf8 -*-

# CONVERSION D'UN NOMBRE n EN BINAIRE
# AVEC AFFICHAGE PAR GROUPE DE QUATRE BITS
# P.G
#
# Amélioration du script de cours02.pdf
# http://pascal.delahaye1.free.fr/


n = input("Entrez la valeur de l'entier naturel à convertir : ")
n = int(n)
b = 2

#  Initialisation des  variables
Q  =  n     #  Q  permet  de  stocker  la  suite  des  quotients
L  =  []    #  La  liste  L  contiendra  les  coefficients de  n  en  base  b
chiffreHexa = ['0','1']


#  Boucle  tant  que  "le  quotient  Q  n'est  pas  nul"
while  Q  !=  0  :
    L.append(Q%b)       #  on  ajoute  le  reste  à  la  liste
    Q  =  Q//b          #  on  calcule  le  nouveau  quotient

#  On  inverse  la  liste  et  on  affiche  le  résultat
L.reverse()

# Affichage du résultat
print('Conversion du nombre : ',n)
print('En base 2')
print(n,': ',end='')
rang=len(L)
for chiffre in L:
    if rang % 4 == 0:
        print(' ',end='')
    rang = rang - 1
    print(chiffreHexa[chiffre],end='')
print(' ',end='\n')
```



Ecriture n en base 2.py

[1] Pour ce chapitre sources consultées :
http://pascal.delahaye1.free.fr/
guillaume.revy@univ-perp.fr C1-ArithmetiqueOrdinateurs.pdf

b) Le codage des entiers relatifs

Le codage est donné sur un nombre de bits déterminé appelé le format. Les nombres négatifs sont représentés par le complément à 2, ou complément vrai. Les nombres négatifs ont la propriété d'avoir le bit de poids fort égal à 1.

Au décodage si le bit de poids fort est à 1 c'est un nombre négatif et pour connaitre sa valeur absolue il suffit de prendre le complément à 2 du nombre.

!! Dans cette représentation tout bit qui apparaît au-delà du format est ignoré.

!! Soustraire un nombre revient à additionner son complément à 2.

c) Exercice sur les nombres entiers

Q1.2 : Coder dans un format de 8 bits les nombres : 44, -103, 72

Q1.3 : Les nombres entiers standards sont codés dans les ordinateurs avec 32 bits.
Calculer les valeurs possibles :
a) Quand le nombre est de type 'unsigned' c'est-à-dire uniquement positif,
b) Quand le nombre n'est pas de type 'unsigned' il peut alors être positif ou négatif.

1.2 Le codage de la partie fractionnaire

$$0.a_{-1}a_{-2}a_{-3}....$$

Est interprété de la manière suivante :

$$0.a_{-1}2^{-1}+a_{-2}2^{-2}+a_{-3}2^{-3}....$$

Les coefficients a_{-1} a_{-2} a_{-3} sont obtenus par la méthode des multiplications successives.

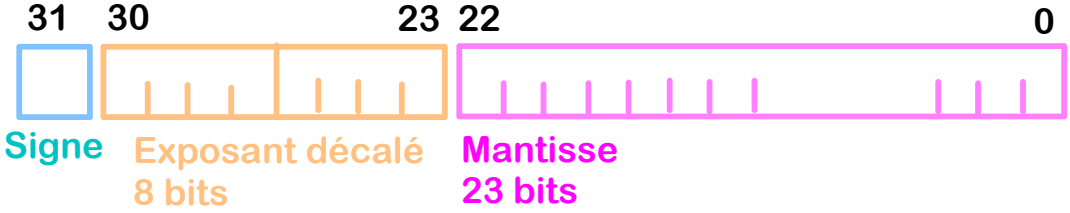
Q1.4 : Déterminer la représentation des nombres suivants **on se limitera à 8 bits maximum** :
a) 0.241
b) 0.625
c) 0.8546

Q1.5 : Coder le script qui réalise le calcul de la partie fractionnaire en base 2 en vous inspirant de la question Q1.1.

1.3 Le codage des nombres flottants avec le format IEEE 754

a) Présentation du codage sur 32 bits

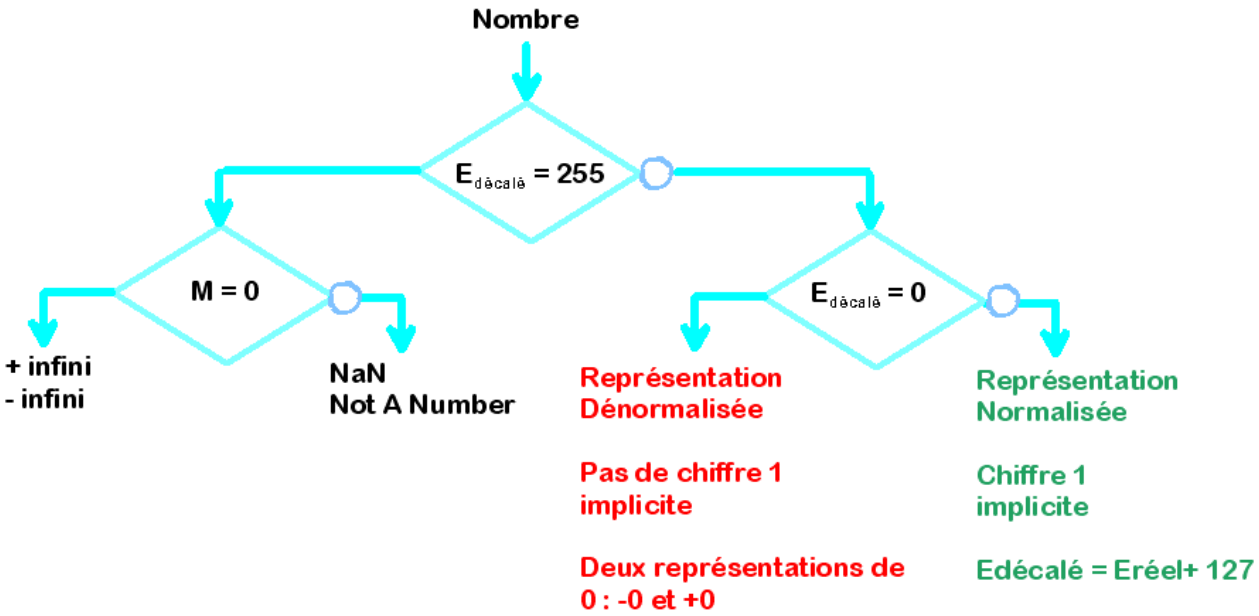
Le codage des nombres flottants est un peu plus complexe, il fait l'objet d'une nombreuse littérature sur internet. Dans ce paragraphe nous verront le codage des nombres flottants en simple précision à savoir un codage sur 32 bits :



Il y a deux catégories de codage le codage normalisé et le codage dénormalisé. La forme générale peut être écrite sous la forme :

$$(-1)^S\ (E_{décalé})\ (\text{Mantisse M})$$

Pour s'y retrouver et savoir à quelle forme on a affaire voir l'algorithme ci-dessous :



premier envol. . . et premier plongeon d'Ariane 5

Premier vol d'Ariane 5 (4 juin 1996)

- Après 39 sec. de vol : autodestruction de la fusée
  - coût de la fusée / du cargot ≈ 500 millions $
  - coût du développement ≈ 7 milliard $
- Système de Référence Inertielle (SRI) : calcule la position, la vitesse et l'inclinaison de la fusée, en fonction de mesures d'accélération et de rotation
  - identique à celui d'Ariane 4
  - accélération 5 fois plus élevée
- Forte accélération de la fusée ⇝ dépassement de capacité lors du calcul des position et vitesse
  - dû à la conversion d'un nombre virgule flottante 64 bits (`double`) en nombre entier de 16 bits dans un logiciel en Ada

Source : Représentation des nombres en machine Guillaume Revy

b) Utilisation de la forme normalisée exemple de codage

Dans cette forme l'exposant décalé varie entre 1 à 254, les valeurs 0 et 255 sont réservées, voir ci-dessus. L'exposant réel varie alors entre 1-127=-126 et 254-127=+127.

$$E_{réél} = E_{décalé} - 127$$

Travaillons cette forme avec un exemple, le codage de +40.

Le signe est positif donc S = 0

Le codage de 40 : 101000

On normalise la mantisse en laissant à gauche de la virgule le chiffre 1 de poids le plus élevé et on note de combien il faut décaler la virgule pour y arriver :

1,01000 décalage de 5 positions vers la droite de la virgule donc on multiplie le nombre par 2^5

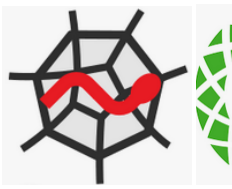
$40 = 1{,}01 \cdot 2^5$

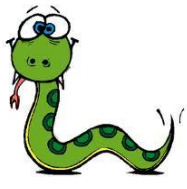
L'exposant réel est égal à 5 donc l'exposant décalé est égal à $E_{décalé} = 5 + 127 = 132$

Codage de $E_{décalé}$: 1000 0100

Codage final du nombre : 0 1000 0100 010----------0

Q1.6 : Coder le nombre - 87,375




c) Utilisation de la forme normalisée exemple de décodage

Décodons le nombre précédent : 0 1000 0100 0100 0000 0000 0000 0000 000

Le bit de signe est à zéro le nombre est positif

L'exposant décalé vaut 1000 0100 soit 132, l'exposant réel vaut 132 - 127 = 5

La mantisse normalisée vaut **1**,01 (on ajoute le 1 implicite de la forme normalisée)

Le nombre vaut donc : + 1.01 • 2^5 = 40

Q1.7 : Décoder le nombre 3EA8000

Missile Patriote (25 février 1991)

- Échec lors de l'interception d'un Scud (Dharan, Arabie Saoudite)
  - bilan : 28 morts / 100 blessés
- Compteur dans la batterie du missile : ajout de 1/10 tous les dixièmes de seconde
  - 1/10 ⇝ non représentable exactement en machine
  - $1/10 \approx (0.0001100110011001100110011001100110011001\ldots)_2$
  - erreur (24 bits) $\approx 9.5 \times 10^{-8}$ par ajout de 1/10
  - au bout de 100h : erreur ≈ 0.34 secondes
- Vitesse du missile Scud : 1676 m/s
  - au bout de 100h : erreur ≈ 568 m

Source : Représentation des nombres en machine Guillaume Revy

Étendue des nombres en format normalisé simple précision

Compléments

Q1.8 : Déterminer le plus petit nombre M_{min} possible dans ce format. Pour ce nombre l'exposant décalé vaut 1 et la mantisse **1**,0.

Q1.9 : Déterminer le plus grand nombre M_{max} possible dans ce format. Pour ce nombre l'exposant décalé vaut 254 et la mantisse **1**,111 1111 1111 1111 1111 1111.

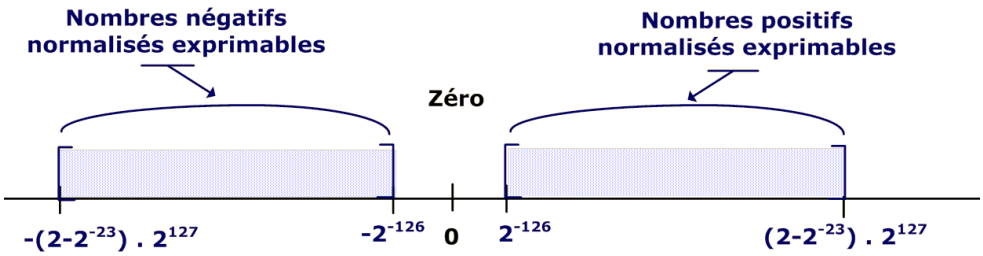
Pour ce calcul on considèrera que la valeur numérique de **1**,111 1111 1111 1111 1111 1111 vaut : $2 - 2^{-23}$

Pour les courageux pour vérifier ce résultat :

Propriété : n est un entier naturel non nul et q un réel différent de 1 alors on a :

$$1+q+q^2+...+q^n=\frac{1-q^{n+1}}{1-q}$$

Remarque : Il s'agit de la somme des n+1 premiers termes d'une suite géométrique de raison q et de premier terme 1.



Novembre 1998, navire américain USS Yorktown, on a par erreur tapé un « zéro » sur un clavier → division par 0. Ce problème n'était pas prévu → cascade d'erreurs → arrêt du système de propulsion.

Les valeurs spéciales dans le format simple précision

Compléments

Quelques tableaux pour résumer[2]

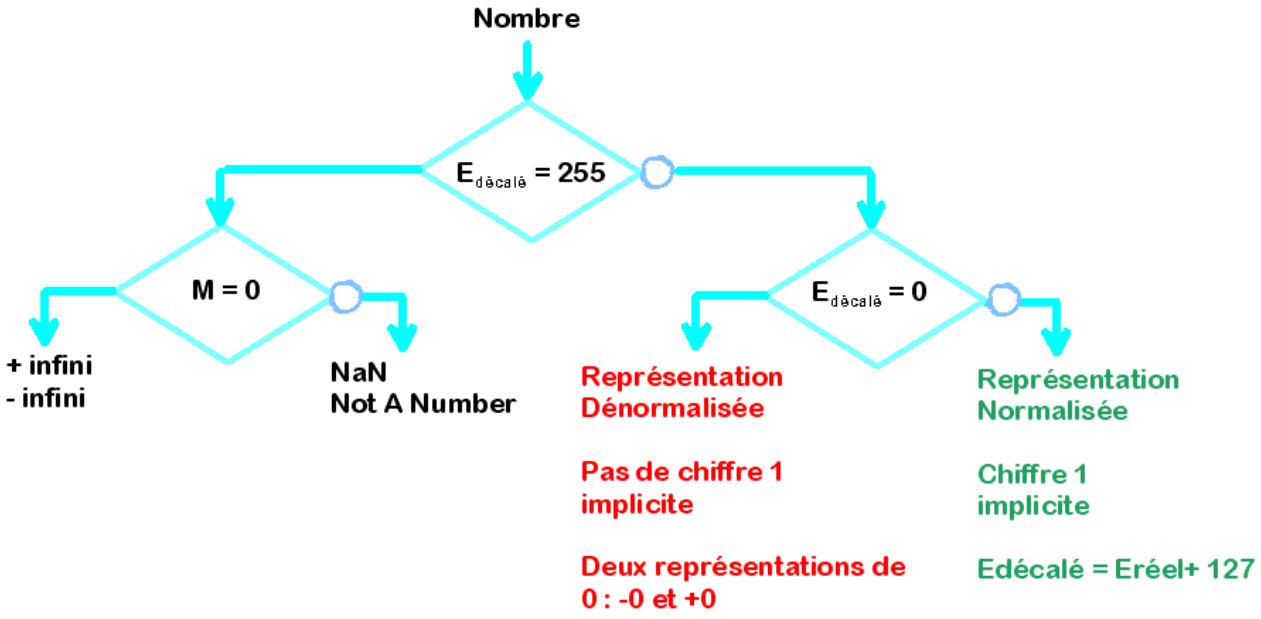
- Encodage des valeurs +0 et −0

+0	0 00000000 00000000000000000000000
−0	1 00000000 00000000000000000000000

- Encodage des valeurs +∞, −∞ et NaN (*Not a Number*)

+∞	0 11111111 00000000000000000000000
−∞	1 11111111 00000000000000000000000
sNaN	1 11111111 00000000000000000000001
qNaN	1 11111111 10000000000000000000001

  - plusieurs représentations pour les deux NaN



[2] D'après Guillaume Revy 01-ualitéNumériqueLogiciel.pdf

2 Les nombres sont-ils dangereux ?

2.1 Pour enquêter sur ce problème effectuons quelques calculs :

Q2.1 : Coder un script en python et réaliser les quatre calculs proposés. Noter les résultats dans le document réponse.

$$\sum_{i=1}^{1000} 0.5 = 500 \qquad \sum_{i=1}^{1000} 0.25 = 250 \qquad \sum_{i=1}^{1000} 0.1 = 100 \qquad \sum_{i=1}^{1000} 0.7 = 700$$

Q2.2 : Quelles observations immédiates pouvez-vous faire au vu de ces résultats ?

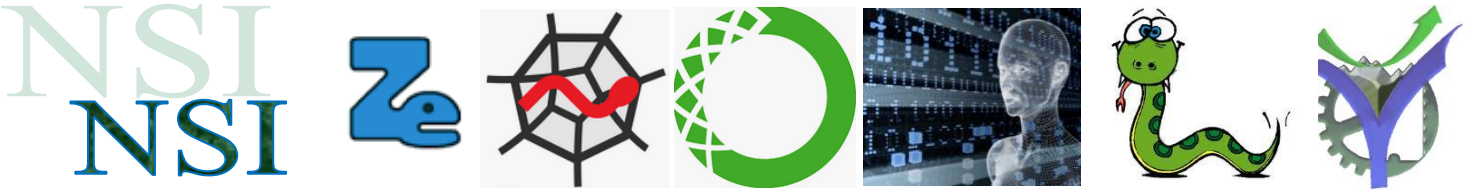
- Guerre du Golfe de 1991 : un anti-missile US Patriot dont le programme tournait depuis 100 heures a raté l'interception d'un missile Irakien Scud - 28 morts
- Explication :
  - l'anti missile Patriot incrémentait un compteur toutes les 0.1 secondes
  - 0.1 approché avec erreur 0.0000000953 (codé sur 24 bits)
  - au bout de 100 heures, erreur cumulée 0.34s
  - dans ce laps de temps le Scud parcourt 500 mètres.

Les résultats obtenus avec deux codes mathématiquement équivalents peuvent donner des résultats différents.

Q2.3 : Coder un script qui réalise le calcul de ces deux sommes et noter les résultats obtenus sur le document réponse :

$$\sum_{i=1}^{n} 1/i \qquad \sum_{i=n}^{1} 1/i$$

Q2.4 : Quelles observations immédiates pouvez-vous faire au vu de ces résultats ?



2.2 Les gros mangent les petits ou l'erreur d'absorption

✍ Q2.5 : Additionner 1 000 000 000 000 000 et 0.01171875 donner le résultat, qu'observez-vous ?

2.3 Vérification des règles de bases[3]

✍ Q2.6 : Effectuer dans la console de python les opérations suivantes et noter les résultats :

a) 0.1 + 0.1 + 0.1
b) 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
c) (0.1 + 0.1 + 0.1) + (0.1 + 0.1 + 0.1)
d) (0.1 + 0.2) + 0.3
e) 0.1 + (0.2 + 0.3)
f) (0.1 * 0.2) * 0.3
g) 0.1 * (0.2 * 0.3)
h) 1000 * (0.1 + 0.2)
i) 1000 * 0.1 + 1000 * 0.2

✍ Q2.7 : Rappeler la règle de l'associativité pour l'addition et la multiplication.

✍ Q2.8 : Rappeler la règle de la distributivité entre l'addition et la multiplication.

✍ Q2.9 : Bilan des opérations. Les opérations addition et multiplication sont-elles associatives ? La distributivité entre la multiplication et l'addition est-elle respectée ?

2.4 Les erreurs d'élimination (ou cancellation)

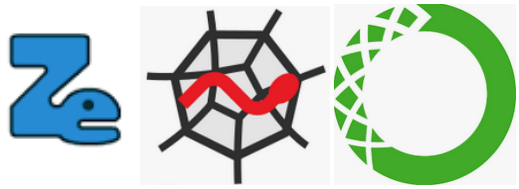
Cette erreur se produit quand on soustrait deux nombres très proches l'un de l'autre.

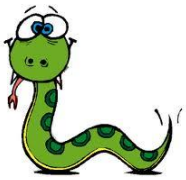
✍ Q2.10 Faire fonctionner le script suivant :

```
# Mise en évidence des erreurs d'élimination
# D'après Eric Obermeyer Représentation des nombres en informatique
a = (1 + 3**-33) - 1
b = 3**(-33)
print("a = ",a)
print("b = ",b)
print("Le résultat théorique a - b est égal à 0")
print("Le résultat calculé est de : ", a - b)
print("Soit une erreur relative de ",(a - b) * 100 / b, " %")
```

De combien est l'erreur relative ?

[3] Pour les points qui suivent merci à Eric Obermeyer Représentation des nombres en informatique






2.5 Les erreurs de comparaison

Ces erreurs se produisent quand on réalise la comparaison entre deux nombres flottants dans un script :

Q2.11 Tester dans la console la comparaison suivante 0.1 + 0.1 + 0.1 == 0.3, quel résultat obtient-on ? Conclusion ?

3 Divertissements. Un investissement rentable[4] ?

Compléments

3.1 Un investissement rentable ?

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\,28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,
- l'année suivante, je prends 1€ de frais et je multiplie par 3,
- ...
- après n ans, je prends 1€ de frais et je multiplie par n,
- Pour récupérer mon argent, il y a 1€ de frais.

Q3.1 Programmer le script qui permet de calculer le gain de cet investissement.

Dans 50 ans, pour ma retraite, combien d'argent aurai-je?

Quel résultat obtenez-vous ?

La valeur exacte est approximativement égale à 0,02 € !!

3.2 Les problèmes d'arrondis à la boulangerie[5]

Après le passage à l'euro en janvier 2002

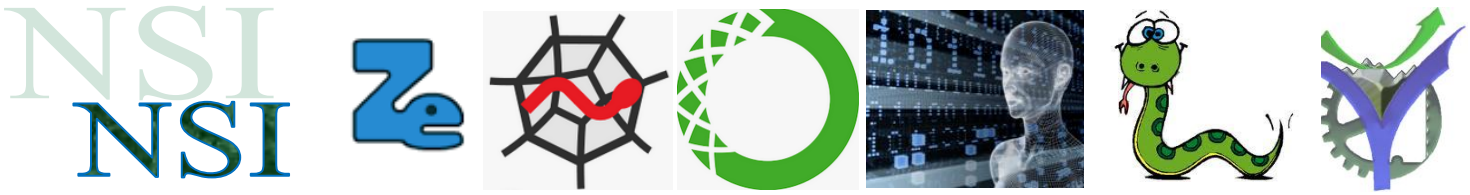
Sans un seul Euro en poche, mais le porte monnaie plein de Francs, un client entre chez sa boulangère préférée pour lui présenter ses meilleurs voeux et acheter une baguette bien croustillante.
1 Euro = 6,5596 Francs et 1 Franc = 0,1524 Euros

C'est combien la baguette maintenant ?
4,30 Francs comme d'habitude

Oui mais à partir d'aujourd'hui c'est en Euros.
Ah, j'oubliais (un petit coup d'EuroCalculette) : ça fait 0,66 Euros.

0,66 Euros (un petit coup d'EuroCalculette)
mais ça fait 4,33 Francs, ma baguette a augmenté de 3 centimes !

[4] D'après Les nombres et l'ordinateur Mme Sylvie Boldo, Boldo2014.pdf p. 205.
[5] D'après Erreurs de calcul des ordinateurs, Mme Jocelyne Erhel, precision-2016.pdf p.13 et ss.



Je vous donne une pièce de 5 Francs et vous me rendez la monnaie en Euros.
Bien, 5 Francs ça fait (EuroCalculette) 0,76 Euros.
Moins 0,66 Euros la baguette, je vous rends 10 centimes d'Euro.

Chouette, ma première pièce en Euro.
Alors (EuroCalculette) 0,10 Euros, cela fait 0,66 Francs, c'est marrant, comme le prix de la baguette en Euros.

Donc je vous ai donné 5 Francs, vous me rendez 0,66 Francs, ça met la baguette à 4,34 Francs ! Le prix a beaucoup augmenté !

Le même client décide le lendemain d'acheter deux baguettes, toujours dans la même boulangerie.

Bonjour, je voudrais deux baguettes. C'est combien en Euros ?
Voyons 4,30 x 2 = 8,60 Francs, soit (EuroCalculette) 1,31 Euros.
Deux baguettes coûtent moins cher que 2 fois une baguette (2 x 0,66), c'est transcendant ce truc !

Je vous donne une pièce de 10 Francs, et comme hier, vous me rendez la monnaie en Euros.
Donc (Eurocalculette) 1,52 Euros moins 1,31 Euros, je vous rends 0,21 Euros.

Voyons voir, (Eurocalculette) 0,21 Euros, cela fait 1,38 Francs, j'ai donc payé (10-1,38)/2 = 8,62/2= 4,31 Francs la baguette.
C'est moins cher qu'hier mais plus qu'en 2001 !

4 Les problèmes d'arrondis[6].

Compléments

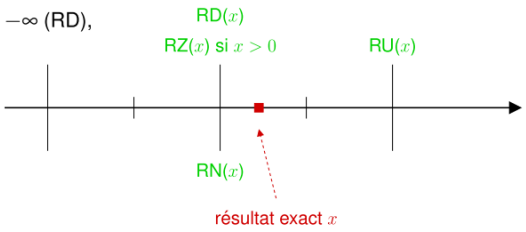
4.1 Les différents arrondis

- Le résultat d'une opération flottante entre deux nombres flottants n'est, en général, pas exactement représentable par un nombre flottant.
  - le résultat doit être arrondi

- La norme IEEE 754 propose 4 modes d'arrondi
  - au plus près pair (RN),
  - vers $+\infty$ (RU), vers $-\infty$ (RD),
  - et vers 0 (RZ)

- Le résultat d'une opération flottante entre deux nombres flottants n'est, en général, pas exactement représentable par un nombre flottant.
  - le résultat doit être arrondi

- La norme IEEE 754 propose 4 modes d'arrondi
  - au plus près pair (RN),
  - vers $+\infty$ (RU), vers $-\infty$ (RD),
  - et vers 0 (RZ)



4.2 Comment arrondir en python

```
1  # -*- coding: utf-8 -*-
2
3  from math import *
4
5  # Entier inférieur (partie entière)
6  arrondiInf = floor(12.3)
7  print(arrondiInf)
8
9  # Entier supérieur
10 arrondiSup = ceil(12.3)
11 print(arrondiSup)
12
13 # Arrondi au plus proche
14 arrondiPro = round(11.7)
15 print(arrondiPro)
16
17 arrondiPro = round(11.5)
18 print(arrondiPro)
19
20 arrondiPro = round(12.5)
21 print(arrondiPro)
```

```
In [3]: (executing lines 1 to 21 of "<tmp 1>")
12
13
12
12
12
```

[6] Ibid.

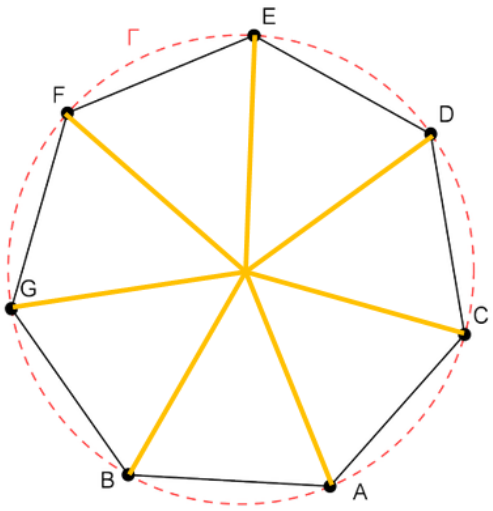
5 Un exemple de problème le calcul approché de pi[7]

Exposons la méthode issue du lien indiqué. En prenant un rayon r = 1 l'aire du cercle est donné par $A = \pi \cdot r^2$ donc $A = \pi$

En calculant la surface comme étant la somme des surfaces de tous les triangles on peut approximer la valeur de π.

En augmentant le nombre de triangles noté n on peut de manière itérative faire le calcul approché de la surface du cercle donc de π.

La démonstration théorique est donnée dans le lien indiqué p1-2.



Algorithme :

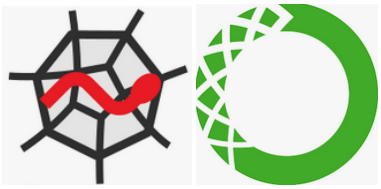
Algorithm 1.1 Algorithme de calcul de π, version naïve

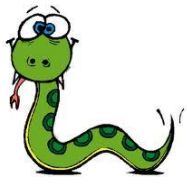
```
1: s ← 1, n ← 4                                   ▷ Initialisations
2: Tantque s > 1e − 10 faire                      ▷ Arrêt si s = sin(α) est petit
3:     s ← sqrt((1 − sqrt(1 − s ∗ s))/2           ▷ nouvelle valeur de sin(α/2)
4:     n ← 2 ∗ n                                  ▷ nouvelle valeur de n
5:     A ← (n/2) ∗ s                              ▷ nouvelle valeur de l'aire du polygône
6: fin Tantque
```

A représente l'aire du cercle donc la valeur approchée de π.

Q5.1 Programmer ce script en python. Quelle valeur de π obtenez-vous ? A partir de quelle valeur de n le résultat ne converge plus vers π.

[7] www.math.univ-paris13.fr/~japhet/Doc/Handouts/RoundOffErrors.pdf






6 Éléments de solution, conduite à tenir, précisions.

6.1 Quelques pistes de solutions

Une bonne approche de ces problèmes de calcul numérique sur ordinateurs est déjà d'y être sensibilisé. Ne pas utiliser le calcul machine en faisant une confiance aveugle dans le résultat obtenu est déjà un premier pas. Quelques pistes lues ça et là.

6.2 Utilisation du module decimal[8]

Ce module permet de travailler avec une plus grande précision

Q6.1 Tester le script ci-dessous :

```
# -*- coding: utf-8 -*-
#  source http://gilles.dubois10.free.fr/Nombres/Reels/machine.html

from math import *
from decimal import *

# calculs grande précision avec le type 'Decimal'.


#avec le type float natif
print()
print("                  2.0 / 3.0 =  %.20f"%(2.0/3.0))

#division grande précision
f=Decimal(2.0)/Decimal(3.0)
getcontext().prec=20

print("Decimal(2.0) / Decimal(3.0) = ",f)
```

6.3 Éviter les erreurs de comparaison

Dans le cas où les deux nombres x et y sont très proches tester l'égalité en utilisant

avec ε = 2^{-51} $\left| 1 - \frac{y}{x} \right| < \varepsilon$

Q6.2 Proposer un script illustrant l'utilité de cette comparaison.

[8] Pour l'utilisation des modules en python voir http://apprendre-python.com/page-python-modules-package-module-cours-debutants-informatique-programmation
et http://apprendre-python.com/page-pip-installer-librairies-automatiquement

7 Pour conclure[9]

On peut prendre toutes les précautions pour réaliser nos calculs numériques mais il ne faut pas oublier que ……..

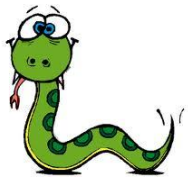
Les erreurs les plus dangereuses sont idiotes

- la sonde Mars Climate Orbiter s'est écrasée sur Mars en 1999 ;
- une partie des développeurs des logiciels supposait que l'unité de mesure était le mètre ;
- l'autre partie croyait que c'était le pied.

⇦

[9] D'après Arithmétique virgule flottante, Jean-Michel Muller Ecole_PRCN_Muller.pdf p.24.




8 Ressources

8.1 La méthode float.hex() sur les nombres flottants

Cette méthode permet d'obtenir la représentation interne du nombre. Le format est explicité ci-dessous.

https://docs.python.org/fr/3/library/stdtypes.html

`float.`**`hex`**()

Donne une représentation d'un nombre à virgule flottante sous forme de chaîne hexadécimale. Pour les nombres à virgule flottante finis, cette représentation comprendra toujours un préfixe `0x`, un suffixe `p`, et un exposant.

classmethod `float.`**`fromhex`**(*s*)

Méthode de classe pour obtenir le *float* représenté par une chaîne de caractères hexadécimale *s*. La chaîne *s* peut contenir des espaces avant et après le chiffre.

Notez que `float.hex()` est une méthode d'instance, alors que `float.fromhex()` est une méthode de classe.

Une chaîne hexadécimale prend la forme :

```
[sign] ['0x'] integer ['.' fraction] ['p' exponent]
```

où `sign` peut être soit `+` soit `-`, `integer` et `fraction` sont des chaînes de chiffres hexadécimales, et `exponent` est un entier décimal facultativement signé. La casse n'est pas significative, et il doit y avoir au moins un chiffre hexadécimal soit dans le nombre entier soit dans la fraction. Cette syntaxe est similaire à la syntaxe spécifiée dans la section 6.4.4.2 de la norme C99, et est aussi la syntaxe utilisée à partir de Java 1.5. En particulier, la sortie de `float.hex()` est utilisable comme valeur hexadécimale à virgule flottante littérale en C ou Java, et des chaînes hexadécimales produites en C via un format `%a` ou Java via `Double.toHexString` sont acceptées par `float.fromhex()`.

Notez que l'exposant est écrit en décimal plutôt qu'en hexadécimal, et qu'il donne la puissance de 2 par lequel multiplier le coefficient. Par exemple, la chaîne hexadécimale `0x3.a7p10` représente le nombre à virgule flottante `(3 + 10./16 + 7./16**2) *2.0**10`, ou `3740.0` :

```
>>> float.fromhex('0x3.a7p10')
3740.0
```

L'application de la conversion inverse à `3740.0` donne une chaîne hexadécimale différente représentant le même nombre :

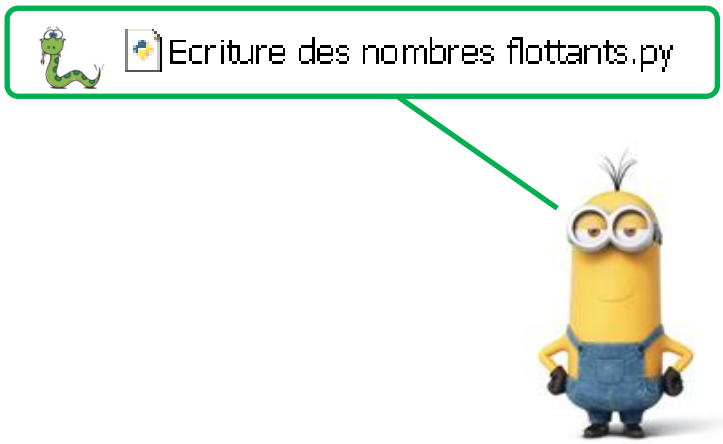
```
>>> float.hex(3740.0)
'0x1.d380000000000p+11'
```

8.2 Affichage complet des nombres flottants

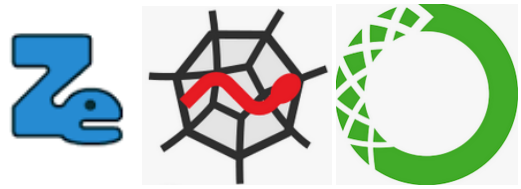
Le code du script ci-dessous permet l'affichage des nombre flottants en simple précision, voir l'exemple d'utilisation ci-dessous :

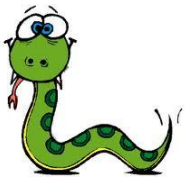
```
Entrez votre nombre : -87.375
-87.375 .hex() =>  -0x1.5d80000000000p+6
-87.375  =>  1.0101110110000000000000000000000000000000000000000000 e 6
-87.375  =>  1 10000000101 0101110110000000000000000000000000000000000000000000
```

```
# -*- coding:Utf8 -*-
#
# ECRITURE DES NOMBRES FLOTTANTS D'après
# http://www.les-mathematiques.net/phorum/read.php?15,887058,1024175
# Revue affichage des réultats P.G
#
# Ecriture des nombres flottants.py

print()
a0 = float(input("Entrez votre nombre réel : " ))
a=a0

if a>0 :
    signe=0
else :
    a=-a
    signe=1
c = 0
b = 1
s = 1
mantisse='1.'
exposant=0
while a>=2:
    a = a/2
    exposant = exposant+1
while a<1:
    a = 2*a
    exposant = exposant-1
a = a-1
```




```
a = a-1
while c>-53:
    c = c-1
    b = b*2
    if b*a >= 1:
        mantisse=mantisse+'1'
        a = a-1/b
        s = s+1/b
    else :
        mantisse=mantisse+'0'

#print(exposant+1023,bin(exposant+1023))
codeexposant=bin(exposant+1023)[2:]
#print(codeexposant)
codeexposant='0'*(11-len(codeexposant))+codeexposant

print(a0,".hex() => ",a0.hex())
print(a0," => ",mantisse,'e',exposant)
print(a0," => ",signe,codeexposant,mantisse[2:])
```






9 Pour mémoire un peu d'orthographe

Compléments

II. Règles d'orthographe

- **Au pluriel, les mots servant à écrire les nombres sont invariables.**

Exceptions

- Les mots « vingt » et « cent » prennent un « s » au pluriel lorsqu'ils ne sont pas suivis d'un autre nombre.

- Les mots « million », « milliard » sont des noms : ils s'accordent au pluriel.

L'écriture des nombres a été simplifiée en 1990 (tirets entre tous les mots servant à écrire des nombres), mais cette nouvelle orthographe est peu utilisée.

Exception

Le trait d'union est parfois remplacé par le mot « et ».

les quatre amis	**trente et une fleurs**
300 = trois-cents	**908 = neuf-cent-huit**
600 000 000 = six-cents-millions	**2000 = deux-mille**
99 = quatre-vingt-dix-neuf	**780 = sept-cent-quatre-vingts**
73 = soixante-treize	**71 = soixante et onze**

45,32 = quarante-cinq virgule trente-deux