TP 18 - Réprésentation des nombres

Commençons par une observation :

➢ Les nombres entiers sont représentés de façon exacte :

```
18 ! = 6402373705728000
19 ! = 121645100408832000
20 ! = 2432902008176640000
25 ! = 15511210043330985984000000
```

➢ Les nombres réels sont représentés sur 64 bits exactement ; ainsi, la représentation est une approximation du nombre.

```
18 ! = 6402373705728000.0
19 ! = 1.21645100408832e+17
20 ! = 2.43290200817664e+18
25 ! = 1.5511210043330986e+25
```

Nombres réels

Mise en contexte :

```
>>> for k in range(1,4):
>>>     print(3*k/10,k/10*3)
0.3 0.30000000000000004
0.6 0.6000000000000001
0.9 0.8999999999999999
```

Le représentation des nombres réels posent plusieurs problèmes dont :

- la fiabilité de la représentation
- l'impact des calculs sur le résultat

Exemple de représentation décimale avec exposant et mantisse :

$$x = 123,456789$$

Son écriture normalisée avec un chiffre à gauche de la virgule est :

$$x = 1,23456789 \times 10^2$$

Son écriture tronquée à 6 chiffres pour la mantisse est :

$$x \approx 1,23456 \times 10^2$$

La représentation est :

signe	mantisse	exposant
0	123456	2

➜ Représentation en binaire

$$(b_k b_{k-1} \cdots b_1 b_0, b_{-1} b_{-2} \cdots b_{-j})_2 = \sum_{i=-j}^{k} b_i 2^i$$

➢ Passage du décimal au binaire des nombres à virgules

- L'écriture de la partie entière s'obtient par la liste renversée des restes dans les divisions successives du nombre par 2.
- Pour la partie fractionnaire, on peut appliquer l'algorithme suivant qui détermine la partie fractionnaire binaire en bornant sa longueur :

```
Entrée : f ∈ [0,1[ et N ∈ ℕ*
L ← [0]
Tant que f > 0 et len(L) ≤ N faire
    Si 2f ≥ 1 alors b ← 1
    Sinon b ← 0
    Insérer b à droite de L
    f ← 2f − b
Fintantque
Sortie : L
```

Exemple : $45,125 = (101101,001)_2 = 2^5 + 2^3 + 2^2 + 2^0 + 2^{-3}$

Exercice 1 Dans cet exercice, on considère des nombres positifs.

1. Par le calcul manuel, déterminer l'écriture décimale de

$$(11100,1011)_2$$

2. Par le calcul manuel, déterminer l'écriture binaire de

$$57,321$$

3. Donner une fonction `binaire(x,n)` qui retourne la chaine de caractère donnant l'écriture binaire de x dont la partie fractionnaire est bornée par n.
4. Donner une fonction `decimal(c)` qui fait l'opération inverse.

➢ Format de la représentation binaire : $s \times "1,m" \times 2^e$

- s est le signe : 0 pour les nombres positifs et 1 pour ceux négatifs
- m la mantisse : $(1, b_{k-1} b_{k-2} \cdots b_{-j})_2$, elle commence par 1 ($b_k \neq 0$) et donc m représente uniquement la partie fractionnaire
- e l'exposant définit la position de la virgule

Convention :

format	signe	mantisse	exposant	total
simple précision	1 bit	23 bits	8 bits	32 bits
double précision	1 bit	52 bits	11 bits	64 bits

En Python les réels sont représentés en *double précision*. L'exposant est donc représenté sur 11 bits pour des valeurs dans $[\![-1024, 1023]\!]$ par l'entier $e + 1024$.

Attention ! Les nombres réels sont représentés par un nombre fini d'écritures, 2^{64} en tout, et donc plusieurs nombres réels correspondent à la même représentation.

➜ Erreurs d'arrondi

⇛ Erreur de représentation en *double précision* :

- Erreur absolue sur la mantisse : $\Delta m \leq 2^{-53}$
- Erreur absolue sur x : $\Delta x = \Delta m \times 2^e$
- Erreur relative sur x : $\dfrac{\Delta x}{|x|} \leq 2^{-53}$

Or $2^{-53} \approx 1{,}11 \times 10^{-16}$, on retient $\dfrac{\Delta x}{|x|} \leq 10^{-15}$.

Attention ! le test d'égalité (==) n'est pas valide sur les réels :

```
>>> 3*0.1==0.3
False
```

L'incertitude du test repose sur l'erreur relative :

```
>>> x=123*1e-100
>>> x==x+1e-17 #erreur absolue
False
>>> x==x+x*1e-17 #erreur relative
True
```

⇛ En pratique, le test d'égalité sur les nombres réels est :

```
abs(a-b)<1e-15
```

```
>>> abs(3*0.1-0.3)<1e-15
True
```

Ceci correspond à une erreur relative associée à l'unité comme ordre de grandeur de référence et non aux grandeurs en présence ! Cela convient pour l'approximation d'une limite, d'une somme ...
Lors d'un travail spécifique, il pourra être pertinent d'utiliser le test suivant fondé sur l'erreur relative associée aux termes considérés : `abs(a-b)<1e-15*max(abs(a),abs(b))`

```
>>> a,b=1.234*1e-30,1.23*1e-31
>>> a-b
1.111e-30 # valeur exacte
>>> abs(a-b)<1e-15
True # erreur relative a 1
>>> abs(a-b)<1e-15*max(a,b)
False # erreur relative a a et b
```

➜ L'*infini* est obtenu au delà du plus grand nombre représentable : $(2^{53}-1)\times 2^{1023-52}$

```
>>> x= (2**53-1)*2.0**(1023-52)
>>> x
1.7976931348623157e+308
>>> x+x*2**(-53)
inf
>>> x+2**(1023-53)
inf
```

➜ Phénomène d'absorption

Lorsque que nous effectuons une addition, c'est le terme qui possède la plus grande imprécision qui détermine la précision du résultat et donc, au final, l'information du termes le plus petit, inférieure à l'imprécision de l'autre, n'est pas prise en compte.

```
>>> for k in range(11,20,2):
>>>     print(k,10**(-k),(1+10**(-k))-1)
11 1e-11 1.000000082740371e-11
13 1e-13 9.992007221626409e-14
15 1e-15 1.1102230246251565e-15
17 1e-17 0.0
19 1e-19 0.0
```

Par exemple, sur une matisse de 8 chiffres (+1) :

$$a = 1{,}00111011 \times 2^2 \text{ et } b = 1{,}11011001 \times 2^{-2}$$

L'imprécision sur a est de l'ordre de 2^{-6} et donc les 4 derniers chiffres de la mantisse de b ne sont pas pris en compte dans le calcul de $a+b$:

	100,	111011	
+	0,	011101	1001
=	101,	011000	

Les conséquences sont fatales dans un calcul : $1 = 0$!

```
>>> for k in range(11,20,2):
>>>  print(k,10**(-k)*10**k,((1+10**(-k))-1)*10**k)
11 0.9999999999999999 1.000000082740371
13 1.0 0.9992007221626409
15 1.0 1.1102230246251565
17 1.0 0.0
19 1.0 0.0
```

Attention ! Additionner des nombres d'ordres de grandeurs différents est source d'erreur

Exercice 2 On sait que

$$\sum_{k=1}^{+\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

Pour $n \in \mathbb{N}^*$ fixé, laquelle des deux sommes suivantes donne la meilleure approximation de la limite ?

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots + \frac{1}{(n-1)^2} + \frac{1}{n^2}$$

$$\frac{1}{n^2} + \frac{1}{(n-1)^2} + \frac{1}{(n-2)^2} + \ldots + \frac{1}{3^2} + \frac{1}{2^2} + 1$$

Exercice 3

1. Montrer que $\left(1+\dfrac{1}{n}\right)^n \to e$
2. Afficher dans un repère logarithmique la suite $\left(\left|\left(1+\dfrac{1}{n}\right)^n - e\right|\right)$ pour $n \in \{10^k, k \in [\![0,20]\!]\}$.
3. Identifier trois phases de la courbes et les expliquer.

➜ Phénomène d'élimination

La différence de termes proches, donc du même ordre de grandeur, opère une simplification des chiffres identiques dans les deux mantisses. Ainsi le nombre de chiffres significatifs de la matisse diminue. Lorsque la simplification recouvre la mantisse, le résultat de la différence n'a plus de pertinence. Illustrons cela grâce à la relation :

$$\frac{1}{x}-\frac{1}{x+1}=\frac{1}{x(x+1)}$$

```
for k in range(14,21):
    x,y=10**k,10**k-1
    print(k,1/y-1/x,1/(x*y))
```

Ce qui donne :

```
14 9.939647405784749e-29 1.00000000000001e-28
15 9.860761315262648e-31 1.000000000000001e-30
16 1.232595164407831e-32 1e-32
17 0.0 1e-34
18 0.0 1e-36
```

Attention ! Éviter de soustraire des nombres proches

➜ Nombre normalisé sur 16bits

A titre d'exercice, nous introduisons un format de représentation de nombre réel à virgule sur 16 bits : $s\times"1,m"\times 2^e$

- $s\in\{0,1\}$ désigne le signe
- m sur 8 bits sans compter la partie entière qui vaut 1 (m est la partie fractionnaire)
- e l'exposant sur 7 bits représenter par $64+e$

Exercice 4

1. Donner l'erreur relative de la représentation d'un réel.
2. Préciser les valeurs possibles de l'exposant.
3. Quel est le plus grand réel qui peut être représenté ?
4. Donner deux fonctions :
   - `base2(n)` qui retourne la liste de chiffre de l'écriture binaire de n,
   - `base10(L)` qui retourne le nombre associé à la liste `L` des chiffres de son écriture binaire.
5. Donner une fonction `bin16(x)` qui retourne la chaine de caratère "sme" représentant x.
6. Donner une fonction `reel16(c)` qui retourne le réel associé à la représentation `c` sur 16bits.