Fonctions et représentation des nombres

D'après un TP de Stéphane Gonnord

Buts du TP

- ☞ Écrire des fonctions réalisant un travail déjà fait sur des exemples.
- ☞ Écrire relativement rapidement des fonctions réalisant des tâches assez simples.
- ☞ Travailler autour de la représentation des nombres en machine
- ☞ Apprendre à « déchiffrer » une fonction simple fournie.
- ☞ Commencer à se frotter à des exemples plus élaborés.

1 Ecrire une fonction

Il s'agit de transformer en fonction des calculs réalisés sur des exemples au TP précédent.... Ce TP (et son corrigé) peuvent être consultés à l'envi... On écrit les fonctions *dans le fichier de script* `.py` ; on sauvegarde et exécute, puis on teste les fonctions en les appelant *dans l'interpréteur*.

EXERCICE 1 *Écrire une fonction prenant en paramètres* (x,n) *et renvoyant* x^n.

```
def puissance(x, n):
    ...
```

Il est bien entendu interdit d'utiliser le calcul direct via `x**n` *!*

```
>>> puissance(3, 17)
129140163
```

EXERCICE 2 *Écrire une fonction prenant en paramètre un entier n et renvoyant* $n! = n(n-1)...3.2.1$.

```
def factorielle(n):
    ...
```

Si tout se passe bien, on aura ça :

```
>>> factorielle(5)
120
```

EXERCICE 3 *Écrire une fonction prenant en paramètres deux entiers a et b avec* $a \leqslant b$, *et renvoyant* $\sum_{k=a}^{b} k^5$.

Par exemple, $\sum_{k=831}^{944} k^5 = 63633265760661375$ *:*

Dans l'exemple suivant, on voit qu'une fonction est en fait un objet comme un autre : il peut être donné en paramètre à une autre fonction !

EXERCICE 4 *Écrire une fonction prenant en paramètre une fonction* f, *deux entiers a et b avec* $a \leqslant b$, *et renvoyant* $\sum_{k=a}^{b} f(k)$. *Exemple : on veut calculer* $\sum_{k=831}^{944} k^{10}$.

```
>>> def f0(x):
        return x**10

>>> somme1(f0, 831, 944)
367241911503651005721610202208 25L
```

EXERCICE 5 *Écrire une fonction prenant en paramètres deux réels, et renvoyant leur maximum.*

```
>>> max2(10, 30), max2(40, 10)
(30, 40)
```

EXERCICE 6 *Écrire une fonction prenant en paramètres trois réels, et renvoyant leur maximum.*

```
>>> max3(10, 30, 40), max3(30, 40, 10), max3(40, 30, 10)
(40, 40, 40)
```

EXERCICE 7 *Écrire une fonction prenant en paramètres trois réels x, y, z, et renvoyant* `True` *si ($x \leqslant y \leqslant z$ ou $z \leqslant y \leqslant x$), et* `False` *sinon.*

```
>>> monotone(10, 15, 20), monotone(15, 10, 20), monotone(20, 15, 15)
(True, False, True)
```

EXERCICE 8 *Écrire une fonction prenant en paramètre un réel M et renvoyant le plus petit entier $n \in \mathbb{N}$ tel que $n^4 - 5n^3 + 4 \geqslant M$.*

```
>>> premier_plus_grand(30), premier_plus_grand(10**10)
(6, 318)
```

EXERCICE 9 *Écrire une fonction prenant en paramètre une fonction f et un réel M, puis renvoyant le plus petit entier $n \in \mathbb{N}$ tel que $f(n) \geqslant M$.*

```
def premier_plus_grand_bis(f, M):
   ...
```

Vérification :

```
>>> def g(x):
        return x**2
>>> premier_plus_grand_bis(g, 10), premier_plus_grand_bis(g, 100),
    premier_plus_grand_bis(g, 1000)
(4, 10, 32)
```

Que va-t-il se passer si un tel n n'existe pas ? Donner un exemple !

EXERCICE 10 *On considère la fonction F qui à tout nombre entier, associe la somme de ses chiffres. Il faut deux itérations à partir de* 19 *pour aboutir à un nombre comportant un seul chiffre.*

En effet, on a : $19 \overset{F}{\longmapsto} 10 \overset{F}{\longmapsto} 1$.

En itérant la fonction F, on aboutit à un nombre avec un seul chiffre appelé racine digitale additive. La persistance additive est le nombre d'itérations nécessaires. Par exemple, la persistance additive de 19 *est de* 2.

1. *Écrire une fonction prenant en paramètre un entier n et renvoyant la somme des décimales de n.*

   Si tout se passe bien, on aura ça :

   ```
   >>> somme_dec(2018), somme_dec(factorielle(100))
   (11, 648)
   ```

2. *Écrire une fonction* `persistance_additive(n)` *qui retourne la persistance additive de l'entier* `n` *passé en paramètre.*

3. *Écrire un programme qui affiche le plus petit entier dont la persistance additive est de* 3 [1].

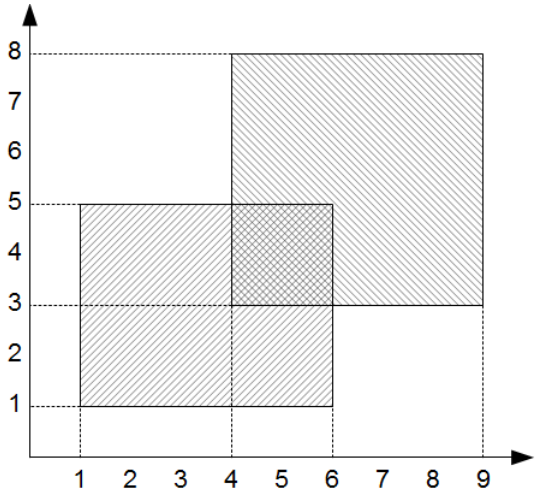
1. 19999999999999999999999 est le plus petit entier dont la persistance additive est 4

EXERCICE 11 1. *Écrire une fonction prenant en paramètres les bornes de deux intervalles fermés* $[a\,;\,b]$ *et* $[c\,;\,d]$ *de* $\mathbb{R}$ *et qui retourne* `True` *si leur intersection est non vide et* `False` *sinon.*

Si tout se passe bien, on aura ça :

```
>>> intersection_intervalle(843, 844, 841, 842)
False
```

2. *Écrire une fonction prenant en paramètres les abscisses minimale et maximale et les ordonnées minimale et maximale de deux rectangles du plan et qui retourne* `True` *si leur intersection est non vide et* `False` *sinon.*



Rappel : 2016 est bissextile, mais pas 2100 (multiple de 100), alors que 2400 l'est (multiple de 400).

EXERCICE 12 *Écrire une fonction renvoyant le caractère bissextile ou non d'une année.*

```
>>> bissextile(2100), bissextile(2400)
(False, True)
```

EXERCICE 13 *Écrire une fonction prenant en paramètre un entier n et renvoyant* `True` *si n est premier et* `False` *sinon.*

Si tout se passe bien, on aura ça :

```
>>> est_premier(17)
True
>>> list(map(est_premier, [5, 91, 2013, 10000000277]))
[True, False, False, True]
```

2 Prévoir le résultat d'une fonction

Pour les trois premiers exercices, on ne se précipite pas sur le clavier : on réfléchit, on répond comme si c'était un écrit d'informatique *sans ordinateur*... et on passe *ensuite* devant le clavier.

EXERCICE 14 *On définit la fonction suivante :*

```
def mystere1(x):
    y = x
    z = y * x
    y = x + y + z
    z = z - y
    return y - z
```

Quelle sera la valeur renvoyée par la commande suivante ?

```
>>> mystere1(10)
```

Et d'une manière générale, quelle est la valeur renvoyée par `mystere1(x)` *?*

EXERCICE 15 *On définit la fonction suivante :*

```
def mystere2(x, y):
    z = x + y
    if z + y > 2 * x:
        return 1
    return -1
```

Quelle sera la valeur renvoyée par la commande suivante ?

```
>>> mystere2(6, 12), mystere2(15, 7)
```

Et d'une manière générale, quelle est la valeur renvoyée par `mystere2(x, y)` *?*

EXERCICE 16 *On définit la fonction suivante :*

```
def mystere3(x, y):
    z = x + y
    if z > y - x:
        w = 10
    else :
        w = y - x
    t = w - z
    if t <= 841:
        return y - z
```

Quelle sera la valeur renvoyée par la commande suivante ?

```
>>> mystere3(-400, 10) , mystere3(-430, 50), mystere3(20, 1000), mystere3(20, -1000)
```

Et d'une manière générale, quelle est la valeur renvoyée par `mystere3(x, y)` *?*

3 Programmer une suite récurrente

EXERCICE 17 Suites de la forme $u_{n+1} = f(n, u_n)$

On considère la suite définie par : $u_0 = 1$ *et* $\forall n \in \mathbb{N},\ u_{n+1} = \sqrt{u_n^2 + \frac{1}{2^n}}$.

1. *Écrire un programme donnant le terme de rang n de cette suite. (On testera ce programme pour* $n = 10$ *et on devra trouver* $u_{10} = 1,7314869$.*)*
2. *Testez le programme pour de grandes valeurs de n. Que peut-on conjecturer ? (Élever le résultat trouvé au carré pour avoir une idée de qui est ce nombre)*
3. *Écrire un programme qui donne le plus petit entier n tel que* $|u_n - \ell| \leqslant \varepsilon$*, où , où* ε *est un réel strictement positif entré par l'utilisateur.*

EXERCICE 18 Suites de la forme $u_{n+2} = f(n, u_n, u_{n+1})$

Compléter la fonction prenant en paramètre un entier n et renvoyant f_n*, avec* $f_0 = 0$*,* $f_1 = 1$*, et pour tout* $n \in \mathbb{N}$*,*

$f_{n+2} = f_n + f_{n+1}$ *(suite de Fibonacci).*

```
def fibonacci(n):
    t = [0, 1]
    for k in range(n):
        t[0], t[1] = ....
    return ....
```

Si tout se passe bien, on aura :

```
>>> fibonacci(10)
55
```

EXERCICE 19 Suites de la forme $u_{n+p} = f(n, u_{n+p-1}, \dots, u_n)$ (récurrentes d'ordre p)

Écrire une fonction prenant en paramètre un entier n et renvoyant le terme u_n de la suite définie par

$$u_0 = 8 \qquad u_1 = 4 \qquad u_2 = 3 \qquad u_{n+3} = u_n u_{n+1} + (n-1)u_{n+2}$$

```
def suite_exo19(n):
    t = [8, 4, 3]
    for k in range(n):
        ....
    return ....
```

```
>>> [suite_exo19(n) for n in range(5)]
[8, 4, 3, 29, 12]
```

EXERCICE 20 Suites mutuellement récurrentes

Soient x et y deux réels positifs. On définit les suites (a_n) et (b_n) par

$$a_0 = x, \quad b_0 = y, \text{ et } \begin{cases} a_{n+1} = \frac{a_n + b_n}{2} \\ b_{n+1} = \sqrt{a_n b_n} \end{cases}.$$

On admet que ces deux suites convergent (en étant adjacentes) vers la même valeur, appelée moyenne arithmético-géométrique de x et y.

1. *Écrire une fonction prenant en paramètres un entier n et deux nombres x et y et renvoyant a_n et b_n.*
2. *Écrire une fonction permettant de calculer une valeur approchée de la moyenne arithmético-géométrique de deux réels positifs à une précision donnée (on utilisera la fonction* `abs`*).*
3. *Avec les conditions initiales $a_0 = 1$ et $b_0 = \frac{1}{\sqrt{2}}$. On admet que $\frac{2a_{n+1}^2}{1 - \sum_{k=0}^{n} 2^k (a_k^2 - b_k^2)} \underset{n \to +\infty}{\longrightarrow} \pi$ (formule de Brent-Salamin)*

   Vérifier numériquement ce résultat.

4 Changement de base pour un entier non signé

On décrit ci-dessous la méthode de conversion en base 2 de l'écriture de l'entier naturel $87 = 8 \times 10^1 + 7 \times 10^0$ *en base* 10.

Méthode *Conversion en base 2*

Pour écrire les entiers naturels en base deux, on utilise deux chiffres : 0 et 1. Si on se donne un entier naturel n, on imagine qu'il représente n objets et on les groupe par paquets de deux, puis on groupe ces paquets en paquets de deux paquets, ...Ainsi,on fait une succession de divisions par 2, jusqu'à obtenir un quotient égal à 0 et l'écriture en base 2 de n est la liste des restes successifs obtenus (0 ou 1, appelés bits) mais dans l'ordre inverse (le premier bit obtenu est le bit de poids faible et le dernier celui de poids fort).

Avec cet *algorithme des divisions en cascades* voici l'exemple de la conversion du nombre décimal 87 en base 2 :

Etape	N (dividende)	Q (Quotient)	R (Reste)
$87 = 2 \times 43 + 1$	87	43	1
$43 = 2 \times 21 + 1$	43	21	1
$21 = 2 \times 10 + 1$	21	10	1
$10 = 5 \times 2 + 0$	10	5	0
$5 = 2 \times 2 + 1$	5	2	1
$2 = 1 \times 2 + 0$	2	1	0
$1 = 0 \times 2 + 1$	1	0	1

On peut écrire :

$$87 = 2 \times (2 \times (2 \times (2 \times (2 \times (2 \times 1 + 0) + 1) + 0) + 1) + 1) + 1$$
$$87 = 2^6 \times 1 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1$$

Ainsi l'écriture de 87 en base 2 est 1010111.

En Python, la primitive `bin` retourne l'écriture binaire d'un en entier en écriture décimale, sous la forme d'une chaîne de caractères préfixée par '0b'.

```
>>> bin(87)
'0b1010111'
>>> bits = list(map(int, bin(87)[2:]))
[1, 0, 1, 0, 1, 1, 1]
>>> n = 0
```

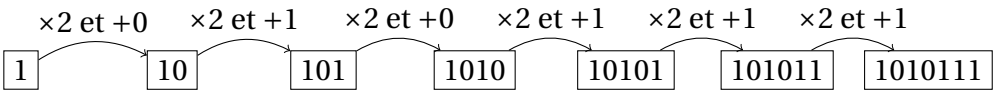
Pour passer d'un tableau contenant l'écriture binaire d'un entier à son écriture décimale, il faut d'abord déterminer si les bits sont rangés dans le tableau par poids décroissant ou croissant. Le sens habituel, choisi pour la fonction `bin` par exemple, est de commencer à l'index 0 du tableau par le bit de poids fort, puis de faire décroître le poids lorsque l'index augmente. Ensuite on peut procéder selon deux algorithmes :

☞ On multiplie chaque bit par la puissance de 2 correspondant à sa position dans le tableau et on somme : à partir de l'écriture binaire `[1, 0, 1, 0, 1, 1, 1]` on retrouve ainsi l'écriture décimale 87. Le premier bit du tableau étant le bit de poids fort, si on appelle ℓ la longueur du tableau, il correspond à $2^{\ell-1}$. Ici $\ell = 7$.

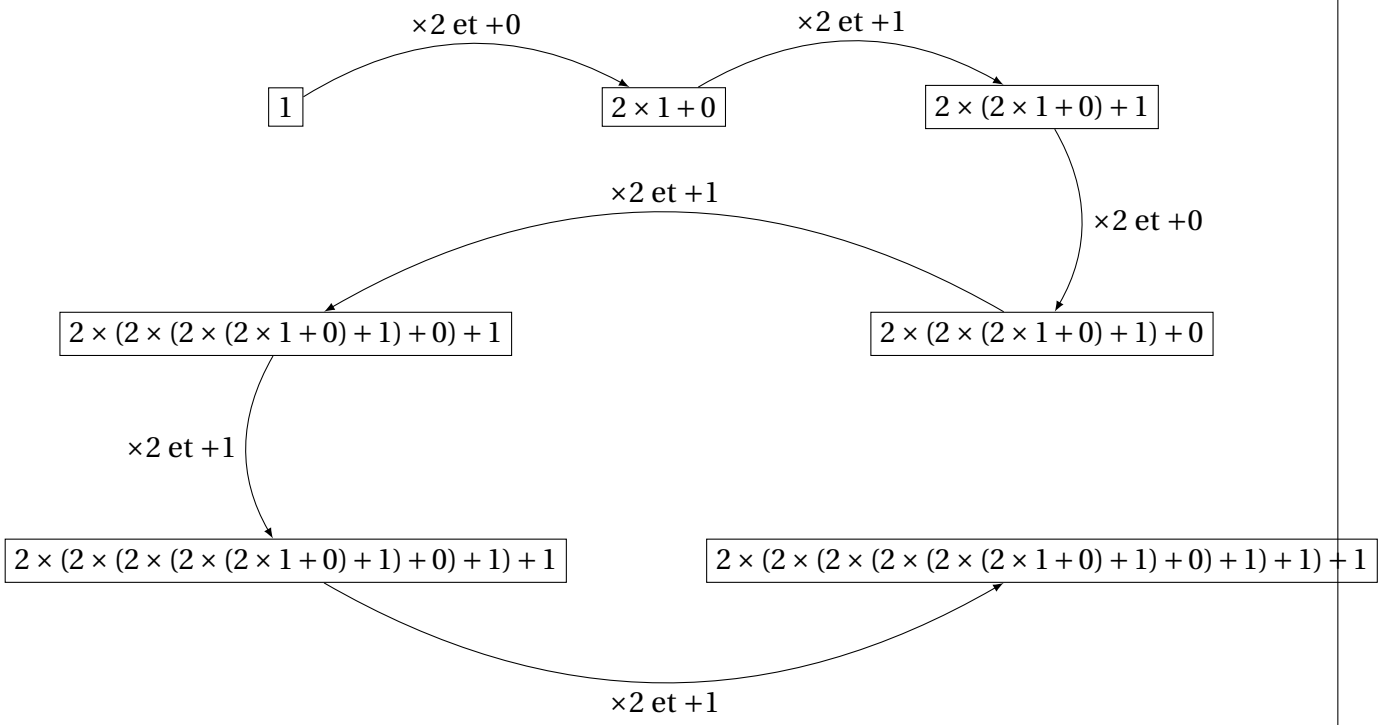
$$87 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

```
>>> for i in range(len(bits)):
...     n += bits[i]*2**(len(bits) - 1 - i)
...
>>> n
87
```

☞ On lit les bits de gauche à droite par poids décroissant, à chaque étape on décale tous les bits du dernier nombre lu vers la gauche et on rajoute le nouveau bit comme bit d'unité. En écriture binaire, un décalage vers la gauche équivaut à une multiplication par 2 :



Il suffit de traduire cet algorithme pour retrouver progressivement l'écriture décimale :



On reconstruit ainsi dans l'autre sens l'expression traduisant la conversion en binaire de 87 par *l'algorithme des divisions en cascades*. On peut lire l'écriture binaire sur la partie droite et la plus grande puissance de 2 inférieure au nombre dans la partie gauche.

Cet algorithme coûte moins de multiplications que le précédent, on n'a pas besoin de calculer les puissances de 2 successives. Il présente donc une meilleure *complexité temporelle*. C'est la réciproque de *l'algorithme des divisions en cascades*.

Ces deux algorithmes de conversion d'écriture de la base 2 en la base 10, peuvent être utilisés pour l'évaluation de polynômes. Le premier est un algorithme naif, le second s'appelle *l'algorithme d'Horner*.

EXERCICE 21 1. *Créer une variable globale* `MAX_BIT` *qui déterminera la taille des tableaux de bits qui seront manipulés. Elle pourra être modifiée selon les besoins.*

2. *Écrire une fonction* `dec_to_bin(n)` *qui prend en argument un entier* `n` *et qui retourne un tableau contenant l'écriture binaire de* `n`*, les bits étant rangés par poids décroissant dans le sens croissant des index. La fonction doit implémenter l'algorithme des divisions en cascades.*

```
In [13]: MAX_BIT = 2

```

```
In [14]: [dec_to_bin(n) for n in range(4)]
Out[14]: [[0, 0], [0, 1], [1, 0], [1, 1]]

In [15]: MAX_BIT = 8

In [16]: dec_to_bin(255)
Out[16]: [1, 1, 1, 1, 1, 1, 1, 1]

In [17]: dec_to_bin(256) #depassement de capacite
Out[17]: [0, 0, 0, 0, 0, 0, 0, 0]
```

3. *Si on fixe la valeur de* `MAX_BIT` *à* 31*, quel est le plus grand entier non signé dont on peut obtenir la représentation binaire exacte avec la fonction* `dec_to_bin(n)` *?*

4. *Écrire une fonction* `bin_to_dec(bits)` *qui prend en argument le tableau de bits de la représentation en binaire d'un entier et qui retourne son écriture décimale. La fonction doit implémenter l'algorithme d'Horner.*

```
>>> MAX_BIT = 8
>>> bin_to_dec([0,1,0,1,0,1,1,1])
87
>>> dec_to_bin(87)
[0, 1, 0, 1, 0, 1, 1, 1]
```

5 Représentation des nombres réels sous forme de flottants

EXERCICE 22 Représentation des flottants selon la norme IEE 754 sur 64 bits (double précision)

- Pour représenter en machine les réels, on les approche par un ensemble fini de rationnels de la forme $\frac{a}{2^n}$, appelés flottants et qui sont choisis de telle sorte qu'il y ait autant de flottants entre $2^{-1}=\frac{1}{2}$ et $2^0=1$, qu'entre 1 et $2=2^1$, qu'entre 2 et $4=2^2$ ...

- En machine, les flottants sont représentés selon la norme IEEE 754, par un triplet unique : **(signe, exposant, mantisse)**. C'est une représentation similaire à la notation scientifique mais en base 2.
  Par exemple pour $-12.015625=-2^3\times\left(1+\frac{1}{2}+\frac{1}{2^9}\right)$ le signe est -1, l'exposant est 3 et la mantisse est $\frac{1}{2}+\frac{1}{2^9}$.

  - Le **signe** est codé par un bit de signe s (0 pour positif et 1 pour négatif).
  - L'**exposant** (positif ou négatif) est codé sur n bits (11 bits pour les flottants codés sur 64 bits) par un entier naturel e. Les valeurs extrêmes 0 et 2^n-1 sont réservées et on ne peut utiliser en fait que 2^n-2 valeurs pour coder comprises entre 1 et 2^n-2. L'exposant réel codé par e est $e-(2^{n-1}-1)$.
  - La **mantisse** m est la partie fractionnaire en base 2.
    Si m est codé sur p bits (52 bits pour les flottants codés sur 64 bits) par $b_1 b_2 \cdots b_p$ alors

$$1.m = 1+\frac{b_1}{2}+\frac{b_1}{2^2}+\cdots+\frac{b_p}{2^p}$$

  Un flottant f codé par le triplet (s,e,m) avec e codé sur n bits et tel que $0<e<2^n-1$ est dit normalisé :

$$f=(-1)^s\times 2^{e-(2^{n-1}-1)}\times(1.m) \tag{1}$$

1. *Pour déterminer la représentation approchée du décimal* $12,625$ *dans le monde des flottants sur* 64 *bits, on peut importer le module* `decimal` *et utiliser la fonction* `Decimal.from_float()` *qui donne la représentation décimale d'un flottant (avec un nombre digits fixé par* `getcontext().prec`*) qui est utilisée par la machine :*

```
>>> from decimal import *
>>> getcontext().prec = 24
>>> Decimal.from_float(12.65)
```

2. *Quels sont le prédécesseur et le successeur immédiat de* 1.0 *parmi les flottants ?*

3. *Que permet de déterminer le script ci-dessous ?*

```
k = 0
while 1. + 10**(-k) > 1:
    k += 1
print('1 + 10**(-%d) == 1 --> True'%k)
```

4. *Quel est le plus grand flottant représentable sur 64 bits ?*

5. *Mathématiquement, l'égalité* $\frac{1}{x(x+1)} = \frac{1}{x} - \frac{1}{x+1}$ *est vraie pour tout réel différent de* -1 *et* 0*. Lorsque* x *devient grand,* $\frac{1}{x}$ *et* $\frac{1}{x+1}$ *deviennent très proches.*
Compléter le script ci-dessous pour qu'il calcule l'erreur absolue $\left| valeur_{réelle} - valeur_{flottante} \right|$ *et* l'erreur relative $\frac{\left| valeur_{réelle} - valeur_{flottante} \right|}{\left| valeur_{réelle} \right|}$ *commises en calculant dans le monde des flottants* $\frac{1}{x(x+1)}$ *ou* $\frac{1}{x} - \frac{1}{x+1}$.

```
from decimal import *
getcontext (). prec =50  # précision en décimal

def approx1(x):
    return 1./x - 1./(x+1)

def approx2(x):
    return 1./(x*(x+1))

def exacte(x):
    return 1/(Decimal(str(x))*(Decimal(str(x))+1))

def erreur_absolue(exacte, approx):
    approxdec = Decimal(approx)
    ......

def erreur_relative(exacte, approx):
    approxdec = Decimal(approx)
    .....
```

Voici ce qu'on obtient lorsqu'on compare les erreurs absolues entre la valeur exacte[2] *et l'approximation comme flottant de* $\frac{1}{x} - \frac{1}{x+1}$ *avec la somme des erreurs absolues pour les approximations de* $\frac{1}{x}$ *et* $\frac{1}{x+1}$.
Que peut-on conjecturer pour l'erreur absolue commise sur une somme (ou une différence) de flottants ?

2. Ou presque avec une précision réglable par exemple à 50 décimales, alors que le type `float` n'offre que 16 chiffres significatifs (on compte à partir de la première décimale non nulle)

```
In [5]: exacte = Decimal('1')/Decimal('1.1') - Decimal('1')/(Decimal('1.1')+1)
In [6]: erreur_absolue(exacte, 1/1.1-1/(1+1.1))
Out[6]: Decimal('3.8449282238100659408610620539941 8E-18')
In [11]: erreur_absolue(Decimal('1')/Decimal('1.1'), 1/1.1) + erreur_absolue(Decimal('1
Out[11]: Decimal('5.6712691301198472627700665296414202 50E-17')
```

On a comparé les erreurs absolues et relatives commises pour ces deux calculs lorsque x parcourt le tableau
`[1.1*2**i for i in range(0, 55, 2)]`. *Interprétez les résultats obtenus et en particulier les évolutions de l'erreur relative*[3] *:*

```
x          |  1/x-1/(x+1)   |     exacte     | erreur absolue |erreur relative
1.1*2**(0) |4.3290043290e-01|4.3290043290e-1 |3.8449282238e-18|8.8817841970e-18
1.1*2**(2) |4.2087542088e-02|4.2087542088e-2 |2.7101403800e-18|6.4392935428e-17
1.1*2**(4) |3.0547409580e-03|3.0547409580e-3 |4.2257389073e-18|1.3833378887e-15
.....................................................................
1.1*2**(46)|1.6723851191e-28|1.6689949730e-28|3.3901460390e-31|2.0312500000e-3
1.1*2**(48)|1.0649622220e-29|1.0431218581e-29|2.1840363905e-31|2.0937500000e-2
1.1*2**(50)|6.9025329207e-31|6.5195116134e-31|3.8302130729e-32|5.8750000000e-2
1.1*2**(52)|4.9303806576e-32|4.0746947584e-32|8.5568589926e-33|2.1000000000e-1
1.1*2**(54)|0.0000000000e+00|2.5466842240e-33|2.5466842240e-33|1.0000000000e+0
x          |   1/(x(x+1))   |     exacte     | erreur absolue |erreur relative
1.1*2**(0) |4.3290043290e-01|4.3290043290e-1 |1.1486723069e-16|2.6534330289e-16
1.1*2**(2) |4.2087542088e-02|4.2087542088e-2 |1.1167647428e-17|2.6534330289e-16
1.1*2**(4) |3.0547409580e-03|3.0547409580e-3 |5.4475065167e-19|1.7832957333e-16
.....................................................................
1.1*2**(46)|1.6689949730e-28|1.6689949730e-28|1.4015090276e-44|8.3973232408e-17
1.1*2**(48)|1.0431218581e-29|1.0431218581e-29|2.4541147411e-45|2.3526635185e-16
1.1*2**(50)|6.5195116134e-31|6.5195116134e-31|3.6716967651e-47|5.6318586158e-17
1.1*2**(52)|4.0746947584e-32|4.0746947584e-32|4.8528250255e-49|1.1909665173e-17
1.1*2**(54)|2.5466842240e-33|2.5466842240e-33|2.6834477366e-49|1.0537025797e-16
```

EXERCICE 23 *On admet que pour tout réel x la suite* $s_n = \sum_{k=0}^{n} \frac{x^k}{k!}$ *converge vers* $\exp(x)$

1. *Ecrire une fonction de signature* `expo(x, n)` *qui retourne le terme de rang* $n \geqslant 0$ *de la suite* (s_n) *sans utiliser la fonction* `factorial` *du module* `math` *et en effectuant n additions et n multiplications.*

2. *Comparer les valeurs approchées retournées par l'appel* `expo(-30, n)` *pour différentes valeurs de* `n` *(attention au risque d'overflow) et* `exp(-30)` *où* `exp()` *est la fonction de bibliothèque importée du module* `math`*.*

   Pour comprendre le phénomène observé, on rappelle que les flottants sont des représentations approchées des réels en précision finie et que lorsqu'on additionne deux flottants d'ordre de grandeurs très différents, le gros absorbe le petit. Avec ce phénomène d'absorption, l'addition des flottants n'est pas associative :

```
>>> x, y = 1e20, 1e-10
>>> (x+y) - x
0.0
>>> y + (x - x)
1e-10
```

3. Lorsqu'on fait la différence de deux flottants très proches, il se produit une perte de précision relative appelée *cancellation*

EXERCICE 24 *Les fonctions* `somme1` *et* `somme2` *ci-dessous calculent* $\sum_{k=1}^{n} \frac{1}{k^4}$.

Dans `somme2`*, on applique le principe de la photo de classe (on commence par sommer les petits).*

Tester les fonctions pour les entrées $n = 10\,000$ *et* $n = 100\,000$.

On donne $\lim_{n\to+\infty} \sum_{k=1}^{n} \frac{1}{k^4} = \frac{\pi^4}{90}$*, quelle méthode de calcul semble la plus précise ? Donner une interprétation.*

```
def somme1(n):
    s = 0
    for i in range(1,n+1):
        s += 1/i**4
    return s
```

```
def somme2(n):
    s = 0
    for i in range(n+1,0,-1):
        s += 1/i**4
    return s
```

EXERCICE 25 *On considère la fonction* f *dite* transformation du boulanger *de* $[0;1]$ *dans* $[0;1]$ *définie par :*

$$f(x) = 2x \text{ si } x < \frac{1}{2} \text{ et } f(x) = 2(1-x) \text{ sinon.}$$

Pour tout réel $x \in [0;1]$*, on définit la suite* (b_n) *par* $b_0 = x$ *et* $b_{n+1} = f(b_n)$.

1. *Écrire une fonction* `boulanger(x)` *qui retourne l'image de* `x` *par la transformation du boulanger.*
2. *Écrire une fonction* `iterations(x, n)` *qui retourne un tableau avec les* `n` *premier termes de la suite* (b_n) *de premier terme* $b_0 = x$*. On peut s'inspirer du code ci-dessous qui alimente un tableau avec les termes successifs d'une suite récurrente définie par* $u_0 = 2$ *et* $u_{n+1} = u_n^2 + 1$.

```
>>> t = [2]
>>> f = lambda x : x**2 + 1
>>> t.append(f(t[-1]))
>>> t
[2, 5]
```

3. *Déterminer les* 100 *premier termes de la suite de premier terme* $b_0 = 1/3$ *avec* `t1 = iterations(1/3, 100)`*. Comparer avec les résultats obtenus par un calcul à la main.*
4. *Importer le module* `decimal`*, régler la précision à* 50 *décimales exactes et reprendre la question précédente avec* `t2 = iterations(Decimal('1')/Decimal('3'), 100)`.

6 Pour ceux qui s'ennuient

EXERCICE 26 Exponentiation rapide itérative

Pour calculer x^n *en* $O(\ln n)$ *multiplications, on peut regarder le calcul suivant :*

$$\begin{aligned} x^{42} &= \underbrace{1}_{res}.\underbrace{(x}_{P}\overbrace{)^{42}}^{N} = \underbrace{1}_{res}.\underbrace{(x^2}_{P}\overbrace{)^{21}}^{N} = \underbrace{x^2}_{res}.\underbrace{(x^4}_{P}\overbrace{)^{10}}^{N} \\ &= \underbrace{x^2}_{res}.\underbrace{(x^8}_{P}\overbrace{)^{5}}^{N} = \underbrace{(x^2.x^8)}_{res}.\underbrace{(x^{16}}_{P}\overbrace{)^{2}}^{N} = \underbrace{(x^2.x^8)}_{res}.\underbrace{(x^{32}}_{P}\overbrace{)^{1}}^{N} \end{aligned}$$

Regarder comment les quantités `res`*,* P *et* N *évoluent au cours du temps, et en déduire un algorithme itératif réalisant le calcul de* x^n *de façon efficace.*

EXERCICE 27 Projet Euler n° 56

A googol (10^{100}) *is a massive number : one followed by one-hundred zeros;* 100^{100} *is almost unimaginably large : one followed by two-hundred zeros. Despite their size, the sum of the digits in each number is only 1.*

Considering natural numbers of the form, a^b*, where* $a, b < 100$*, what is the maximum digital sum ?*

7 Besoin d'indications ?

- Exercice 26. Si on a bien compris l'exemple, on trouve l'algorithme suivant, qui se traduit facilement en Python :

  Entrées : x, n
  $res \leftarrow 1$; $N \leftarrow n$; $P \leftarrow x$
  tant que $N > 0$ **faire**
  si N *est impair* **alors**
  $res \leftarrow res * P$
  $N \leftarrow (N-1)/2$
  sinon
  $N \leftarrow N/2$
  $P \leftarrow P^2$
  Résultat : res

  On peut factoriser *une instruction... et prendre une condition de sortie différente (sortir lorsque* $N = 1$*), mais alors attention au cas* $n = 0$*. On peut remarquer aussi que celà revient à écrire en base* 2 *l'exposant* $42 = 2 + 2^3 + 2^5$.