Corrigé du TP 04 Fonctions

Chenevois-Jouhet-Junier-Rebout

1 Écrire une fonction

1.1 Exercice 1

```
def puissance(x,n):
    '''retourne x**n sans l'opérateur **'''
    p, pas = 1, 1
    if n < 0:
        x, pas = 1./x, -1
    for i in range(0, n, pas):
        p *= x
    return p
"""
In [9]: [puissance(2, i) for i in range(-2, 3)]
Out[9]: [0.25, 0.5, 1, 2, 4]
"""
```

1.2 Exercice 2

```
def factorielle(n):
    '''retourne 1x2x3x...x(n-1)xn'''
    if n < 0:
        return None
    f = 1
    for i in range(2,n+1):
        f *= i
    return f

"""
In [19]: [factorielle(i) for i in range(-1, 6)]
Out[19]: [None, 1, 1, 2, 6, 24, 120]
"""
```

1.3 Exercice 3

```
def somme_exo3(a,b):
    '''retourne la somme des puissances de 5 des entiers entre a et b'''
```

```
    if b < a:
        a, b = b, a
    s = 0
    for k in range(a, b+1):
        s += k**5
    return s

"""
In [26]: somme_exo3(831, 944), somme_exo3(944, 831)
Out[26]: (63633265760661375, 63633265760661375)
"""
```

1.4 Exercice 4

```
def somme_exo4(a, b, f):
    '''retourne la somme des f(k) avec k entier entre a et b.
    a et b de type entier et f de type fonction'''
    if b < a:
        a, b = b, a
    s = 0
    for k in range(a, b+1):
        s += f(k)
    return s

"""
In [28]: somme_exo4(1, 100), 100*101/2
Out[28]: (5050, 5050.0)
In [32]: somme_exo4(831, 944, lambda x : x**10)
Out[32]: 367241911503651005721610202208 25
In [33]: sum(map(lambda x : x**10, range(831, 945)))
Out[33]: 367241911503651005721610202208 25
"""
```

Sous Python2 l'entier obtenu dépasse 2**31 - 1, il est de type 'entier long'

```
"""
>>> somme_exo4(831, 944, lambda x : x**10)
367241911503651005721610202208 25L
>>> type(_)
<type 'long'>
"""
```

En Python, pas de condition de type sur les paramètres de la fonction, c'est au développeur d'utiliser des arguments dont le type est attendu, d'où l'importance des `docstrings`.

```
"""
In [6]: somme_exo4(lambda x : x**10, 831, 944)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-59c0682f09a0> in <module>()
----> 1 somme_exo4(lambda x : x**10, 831, 944)
.....................
----> 4     if b < a:
      5         a, b = b, a
      6     s = 0
```

```
TypeError: unorderable types: int() < function()
"""
```

1.5 Exercices 5 et 6

```
def max2(a, b):
    '''retourne le maximum de deuxs entiers a et b'''
    if a >= b:
        return a
    return b

def max3(a, b, c):
    '''retourne le maximum de trois entiers a et b'''
    return max2(max2(a,b),c)

"""
In [39]: max3(10,30,40),max3(30,40,10),max3(40,30,10),max3(10,10,10)
Out[39]: (40, 40, 40, 10)
"""
```

1.6 Exercice 7

```
def monotone(x, y, z):
    return x <= y <= z or z<= y <= x

"""
In [41]: monotone(10, 15, 20), monotone(15, 10, 20), monotone(20, 15, 15)
Out[41]: (True, False, True)
"""
```

1.7 Exercices 8 et 9

```
def premier_plus_grand(M):
    '''retourne le plus petit entier n tel que n**4-5*n**3+4 >= M'''
    n = 0
    while n**4-5*n**3+4 < M:
        n += 1
    return n


def premier_plus_grand_bis(f, M):
    '''prend en entrée une fonction f et un réel M et retourne le plus petit
    entier n tel que f(n)>=M. Boucle infinie si un tel entier n'existe pas. '''
    n = 0
    while f(n) < M :
        n += 1
    return n

"""
```

```
In [43]: g = lambda x : x**2
In [44]: premier_plus_grand_bis(g,10),premier_plus_grand_bis(g,100),
premier_plus_grand_bis(g,1000)
Out[44]: (4, 10, 32)
"""
```

1.8 Exercice 10

```
def somme_dec(n):
    '''retourne la somme des décimales de n'''
    s = 0
    while n > 0:
        decimale = n%10
        s += decimale
        n = n//10
    return s

def persistance_additive(n):
    iteration = 0
    s = somme_dec(n)
    while s != n:
        n = s
        s = somme_dec(n)
        iteration = iteration + 1
    return iteration



n = 1
while  persistance_additive(n) != 3:
    n = n + 1
print(n)
```

Le plus petit entier dont la persistance additive est 3 est 199. A propos des persistances additives et multiplicatives on pourra consulter http://villemin.gerard.free.fr/Wwwgvmm/Addition/RAcNum.htm

1.9 Exercice 11

```
def intersection_intervalle(a, b, c, d):
    '''Caractérise l'intersection de [a,b] et [c,d]'''
    return b >= c and  a <= d

def intersection_rectangle(xmin1, xmax1, ymin1, ymax1,
xmin2, xmax2, ymin2, ymax2):
    '''Caractérise l'intersection de deux rectangles à partir de
    leurs abscisses et ordonnées minimales et maximales'''
    return intersection_intervalle(xmin1, xmax1, xmin2, xmax2) and \
    intersection_intervalle(ymin1, ymax1, ymin2, ymax2)
```

1.10 Exercice 12

```
def bissextile(a):
    return  (a%100 != 0 and a%4 == 0) or (a%400 == 0)

"""
In [46]: bissextile(2100), bissextile(2400)
Out[46]: (False, True)
"""
```

1.11 Exercice 13

```
def est_premier(n):
    '''Test de primalité'''
    if n <= 1:
        return False
    d = 2
    bsup = math.sqrt(n)
    while d <= bsup:
        if n%d == 0:
            return False
        d += 1
    return True

"""
In [49]: list(map(est_premier, [-1, 1, 2, 3, 5, 91, 2013, 10000000277]))
Out[49]: [False, False, True, True, True, False, False, True]
"""
```

2 Prévoir le résultat d'une fonction

2.1 Exercice 14

```
def mystere1(x):
    '''fonction mystere qui retourne x**2+4*x
    mystere1(10)==140'''
    y = x
    z = y*x
    y = x+y+z
    z = z-y
    return y-z
```

2.2 Exercice 15

```
def mystere2(x,y):
    '''mystere2(6,12)==1 car 30>12
    mystere2(15,7)== -1 car 29<30
    retourne 1 si 2y>x et -1 sinon '''
    z = x+y
    if z+y>2*x:
        return 1
```

```
    else:
        return -1
```

2.3 Exercice 16

```
def mystere3(x,y):
    '''Valeur retournée :
    1er cas x>0 : si x>=0 et y>=-831-x retourne -x
                  sinon retourne None
    2ème cas x<=0 : si -2x<=841 retourne -x sinon retourne None
    '''
    z = x+y
    if z>y-x: #équivaut à if x>0
        w = 10
    else:
        w = y-x
    t = w-z # t==10-x-y si x>0 ou t==-2x si x<=0
    if t<= 841:
        return y-z

"""
>>> mystere3(-400,10),mystere3(-430,50),mystere3(20,1000),mystere3(20,-1000)
(400, None, -20, None)
"""
```

3 Programmer une suite récurrente

3.1 Exercice 17

```
def suite_exo17(n):
    '''Calcule le terme de rang n de la suite
    définie par u(0)=1 et u(n)=sqrt(u(n-1)**2+1/2**(n-1)))'''
    if n < 0:
        return None
    u, p = 1, 2
    for k in range(1, n+1):
        p *= 1/2 #mise à jour de 1/2**(k-1)
        u = math.sqrt(u**2+p)
    return u

"""
In [55]: [suite_exo17(k) for k in [10, 100, 1000]]
Out[55]: [1.7314868971493835, 1.7320508075688763, 1.7320508075688763]

In [56]: list(map(lambda x : x**2, _))
Out[56]: [2.998046875, 2.999999999999964, 2.999999999999964]

On  peut conjecturer que la suite u(n)
converge vers sqrt(3)
"""
```

```
def seuil_exo17( epsilon):
    '''Retourne le plus petit entier n
    pour lequel abs(u(n) - sqrt(3)) <= epsilon. u(n) converge vers sqrt(3)'''
    u, n, p = 1, 0, 2
    limite = math.sqrt(3)
    while abs(u - limite) > epsilon:
        p *= 1/2 #mise à jour de 1/2**(k-1)
        n += 1
        u = math.sqrt(u**2+p)
    return n

"""
In [58]: seuil_exo17(0.1)
Out[58]: 3
In [60]: abs(suite_exo17(2) - math.sqrt(3)), abs(suite_exo17(3) - math.sqrt(3))
Out[60]: (0.15091197748468743, 0.07373841239117707)
"""
```

3.2 Exercice 18

```
def fibonacci(n):
    '''retourne le terme de rang n de la suite de fibonacci'''
    if n < 0:
        return None
    t = [0, 1]
    for k in range(n):
             t[0], t[1] = t[1], t[0] + t[1]
    return t[0]

# In [17]: [fibonacci(i) for i in range(-1, 6)]
# Out[17]: [None, 0, 1, 1, 2, 3, 5]
```

3.3 Exercice 19

```
def suite_exo19(n):
    assert n >= 0, 'n doit être positif'
    t = [8, 4, 3]
    for k in range(n):
        t[0], t[1], t[2] = t[1], t[2], t[0]*t[1] + (k - 1)*t[2]
    return t[0]

"""
Un exemple d'utilisation d'une assertion :
In [65]: assert 0.3 == 3*0.1, 'Python ne sait pas compter'
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-65-5a76c83276b5> in <module>()
----> 1 assert 0.3 == 3*0.1, 'Python ne sait pas compter'
AssertionError: Python ne sait pas compter

In [67]: [suite_exo18(n) for n in range(5)]
```

```
Out[67]: [8, 4, 3, 29, 12]
"""
```

3.4 Exercice 20

```
def suite_exo20(n, x, y):
    assert n >= 0, 'n doit être positif'
    t = [x, y]
    for k in range(n):
        t[0], t[1] = (t[0] + t[1])/2., (t[0]*t[1])**(1/2.)
    return t

def suite_exo20_approx(x, y, epsilon):
    a, b = x, y
    while abs(a - b) >= epsilon:
         a, b = (a+b)/2., (a*b)**(1/2.)
    return a, b

def approx_pi(n):
    '''Approximation de pi avec la formule de Brent-Salamin'''
    a, b = 1, 1/2**(1/2.)
    p = 1 #puissance de 2
    somme = 0
    for k in range(1, n+1):
        somme += p*(a**2 - b**2)
        a, b = (a+b)/2., (a*b)**(1/2.)
        p *= 2
    return 2*a**2/(1. - somme)

"""
In [70]: [approx_pi(k) for k in range(5)]
Out[70]:
[2.0,
 2.9142135623730954,
 3.14057925052217,
 3.141592646213547,
 3.1415926535898078]
"""
```

4 Changement de base pour un entier non signé

4.1 Exercice 21

- Création d'une variable globale `MAX_BIT` qu'on initialise à 8 bits.

```
MAX_BIT = 8
```

- Fonction `dec_to_bin(n)` de conversion d'écriture décimale en binaire, retourne un tableau de `MAX_BIT` bits. Il faut noter que si l'entier passé en argument est code sur plus de `MAX_BIT` bits, on a un **dépassement de capacité**.

```
def dec_to_bin(n):
    '''Conversion d'un entier n en binaire par l'algorithme
    des divisions en cascades. Retourne un tableau de MAX_BIT bits'''
    bits = [0]*MAX_BIT
    k = MAX_BIT - 1 #on commence par remplir les bits de poids faible
    while k >= 0 and n > 0:
        bits[k] = n%2
        n = n//2
        k = k - 1
    return bits
"""
In [16]: dec_to_bin(255)
Out[16]: [1, 1, 1, 1, 1, 1, 1, 1]

In [17]: dec_to_bin(256)   #dépassement de capacité sur 8 bits
Out[17]: [0, 0, 0, 0, 0, 0, 0, 0]
"""
```

Deux autres versions de cette fonction où la taille du tableau retourné n'est pas limitée. Attention la boucle doit s'exécuter au moins une fois pour que l'écriture binaire de 0 soit retournée et si on insère les bits à la fin, le tableau doit être renversé car on commence par les bits de poids faibles soit avec `bits.reverse()` soit avec un slicing `bits[::-1]`

```
def dec_to_binV2(n):
    bits = [n%2]
    n = n // 2
    while n > 0:
        bits.append(n%2)
        n = n//2
    bits.reverse()
    return bits

def dec_to_binV3(n):
    bits = []
    while n//2 > 0:
        bits.append(n%2)
        n = n//2
    bits.append(n%2)
    return bits[::-1]

"""
In [21]: [dec_to_binV2(n) for n in range(4)]
Out[21]: [[0], [1], [1, 0], [1, 1]]

In [22]: [dec_to_binV3(n) for n in range(4)]
Out[22]: [[0], [1], [1, 0], [1, 1]]
"""
```

- Le plus grand entier non signé qu'on peut réprésenter sur 31 bits est $2^{31} - 1$.
- Fonction `bin_to_dec(bits)` de conversion d'écriture binaire en décimale

```
def bin_to_dec(bits):
    '''Conversion de l'écriture d'un entier de la base dix à la base deux'''
    n = 0
    for k in range(len(bits)):
```

```
        n = 2*n + bits[k]
    return n
"""
>>> [bin_to_dec(dec_to_bin(k)) for k in range(17)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
"""
```

5 Représentation des nombres réels sous forme de flottants

5.1 Exercice 22

- Question 1, avec import du module `decimal`.

```
from decimal import *
"""
In [2]: getcontext().prec = 24

In [3]: Decimal.from_float(12.65)
Out[3]: Decimal('12.650000000000000355271367880050092935562133789062 5')
"""
```

- Question 2 : $1 = 2^0$ et la mantisse d'un flottant en double précision sur 64 bits est codée sur 52 bits. Le successeur de 1 parmi les flottants est donc $1 + 2^{-52}$

```
"""
In [4]: 1 + 2**(-53) == 1
Out[4]: True
In [5]: 1 + 2**(-52) == 1
Out[5]: False
"""
```

- Question 2 (suite) : pour le prédécesseur de 1, il faut considérer que l'écart minimum entre 2 flottants compris entre 2^{-1} et 2^0 est $2^{-1} \times 2^{-52} = 2^{-53}$. Le prédécesseur de 1 parmi les flottants est donc $1 - 2^{-53}$.

```
"""
In [9]: 1 - 2**(-53) == 1
Out[9]: False

In [10]: 1 - 2**(-54) == 1
Out[10]: True
"""
```

- Question 3

```
k = 0
while 1. + 10**(-k) > 1:
    k += 1
print('1 + 10**(-%d) == 1 --> True'%k)
"""
In [11]: (executing lines 381 to 384 of "correcTP4-Fonctions-Suites-Flottants.py")
1 + 10**(-16) == 1 --> True
"""
```

Ce script permet de déterminer le plus petit entier k tel que $1 + 10^{-k}$ inférieur au plus petit écart avec le flottant successeur de 1 qui est $1 + 2^{-52}$ donc $k > \log(2^{-52})/(-\log(10))$.

```
"""
# In [13]: math.log(2**(-52))/-math.log(10)
# Out[13]: 15.65355977452702
"""
```

On trouve 16 (nombre de chiffres significatifs pour le type `float`). On peut avoir plus d'infos sur le type `float` utilisé par Python avec `sys.float_info`.

```
"""
In [19]: import sys
In [20]: sys.float_info
Out[20]: sys.float_info(max=1.7976931348623157e+308,
 max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308,
 min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
 epsilon=2.220446049250313e-16, radix=2, rounds=1)
"""
```

Si x est un réel et fx sa représentation sous forme de flottant on a $fx = x \times (1 + \varepsilon)$.

ε est *l'erreur relative* et *l'erreur absolue* est $\varepsilon \times x$.

- Question 4 :

  Au format double précision sur 64 bits, l'exposant est codé sur 11 bits. Deux valeurs (0 et $2^{11} - 1$) sont réservées aux valeurs spéciales (`Inf`, `NAN`, 0) et il y a autant d'exposants négatifs que de positifs, donc l'exposant maximal est $(2^{11} - 2)/2 = 2^{10} - 1 = 1023$.

  De plus la mantisse est codée sur 52 bits. Le plus grand flottant est donc $2^{1023} \times (1 + 2^{-1} + \ldots + 2^{-52}) = 2^{1023} \times (1 - 2^{-53})/2^{-1} = 2^{1024} - 2^{971}$.

  Attention pour obtenir l'infini il ne suffit pas d'ajouter 1 à ce nombre, il faut ajouter au moins le + petit écart entre 2 flottants au-delà de 2^{1023} qui est $2^{-52} \times 2^{1023}$.

```
"""
In [48]: maxfloat = sum(2.**(1023-i) for i in range(53))

In [49]: maxfloat == float(2**1024 - 2**971)
Out[49]: True

In [50]: maxfloat + 2**(-52)*2**1023
Out[50]: inf
"""
```

- Question 5

```
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):
    #Attention Decimal(0.1) est moins précis que Decimal(str(0.1))
    return 1/(Decimal(str(x))*(Decimal(str(x))+1))

def erreur_absolue(exacte, approx):
```

```
    approxdec = Decimal(approx)
    return abs(exacte - approxdec)

def erreur_relative(exacte, approx):
    approxdec = Decimal(approx)
    return abs(exacte - approxdec)/abs(exacte)

print("{:<11}|{:^16}|{:^16}|{:^16}|{:^14}".format('x','1/x-1/(x+1)',
'exacte', 'erreur absolue', 'erreur relative'))
for i, x in enumerate([1.1*2**i for i in range(0, 55, 2)]):
    print("{:<11}|{:<16.10e}|{:<16.10e}|{:<16.10e}|{:<16.10e}".format('1'
    '.1*2**(%d)'%(2*i),approx1(x),exacte(x),
    erreur_absolue(exacte(x), approx1(x)),erreur_relative(exacte(x),approx1(x))))

print("{:<11}|{:^16}|{:^16}|{:^16}|{:^14}".format(
'x','1/(x(x+1))', 'exacte','erreur absolue', 'erreur relative'))
for i, x in enumerate([1.1*2**i for i in range(0, 55, 2)]):
    print("{:<11}|{:<16.10e}|{:<16.10e}|{:<16.10e}|{:<16.10e}".format(
    '1.1*2**(%d)'%(2*i),approx2(x),exacte(x),
    erreur_absolue(exacte(x),approx2(x)),erreur_relative(exacte(x),approx2(x))))
"""
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**(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**(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
"""
```

5.2 Exercice 23

La suite définie par $\begin{cases} u_0 = 1/3 \\ u_{n+1} = 4u(n) - 1 \end{cases}$ est constante : $\forall n \in \mathbb{N}$, $u_n = 1/3$.

```
def suiteu(n):
    if n < 0:
        return None
    u = 1/3
    print('u(0)= ', u)
    for k in range(1, n+1):
        u = 4*u - 1
        print('u(%d)=%1.15f'%(k, u))
```

Le développement de $1/3$ en base 2 est $0,01010101\ldots$ (infini et périodique).

Si on avait une infinité de bits en machine $4 \times (1/3) - 1$ redonnerait $0,0101\ldots$ puisqu'en base 2 nmultiplier par $4 = 2^2$ revient à décaler la virgule de 2 rangs vers la droite. Mais au départ la représentation de $1/3$ sous forme de

flottant est $2^{-2} \times 1,0101\ldots$ (26 fois la période 01). Donc en multipliant par 4 on fait remonter chaque bit exact puis on le supprime en enlevant 1. Après 26 applications de la relation de récurrence, il ne devrait plus rester de bits exacts comme on peut le vérifier ci-dessous. A partir de $u(27) = 0$ il ne reste plus de bits exacts.

```
"""
In [58]: suiteu(50)
u(0)=  0.3333333333333333
u(1)=0.333333333333333
u(2)=0.333333333333333
u(3)=0.333333333333332
u(4)=0.333333333333329
......................
u(25)=0.312500000000000
u(26)=0.250000000000000
u(27)=0.000000000000000
u(28)=-1.000000000000000
.......................
u(49)=-5864062014805.000000000000000
u(50)=-23456248059221.000000000000000
"""
```

5.3 Exercice 24

```
import math

def expo(x, n):
    '''Retourne une approximation de exp(x)
    avec la formule exp(x) = 1 + x + ...+x**n/n!'''
    if n >= 0:
        somme, terme = 1, 1
        for k in range(1, n+1):
            terme *= float(x)/k
            somme += terme
        return somme
```

Notre fonction maison fait ce qu'on attend pour les $x > 0$.

```
"""
In [63]: expo(1, 50)
Out[63]: 2.7182818284590455
In [64]: expo(10, 50)
Out[64]: 22026.465794806714
In [66]: math.exp(10)
Out[66]: 22026.465794806718
"""
```

Pour les $x < 0$ petits en valeur absolue tout va bien.

```
"""
In [67]: expo(-1, 50)
Out[67]: 0.36787944117144245

In [68]: math.exp(-1)
Out[68]: 0.36787944117144233
"""
```

Mais si $x < 0$ est plus grand en valeur absolue, dans le calcul de la somme des $x^k/k!$ pour k entre 0 et n, on ajoute des nombres de signes opposés + ou −,on a donc des phénomènes de *cancellation* et aussi des phénomènes d'*absorption* lorsque k assez grand et que $x^k/k!$ devient très petit sans compter qu'il finit par atteindre le zéro machine.

```
x = -30
val = math.exp(x)
for n in range(120):
    print('n : %d Bibliothèque : %1.6e et \
    Maison : %1.6e'%(n, val, expo(x, n)))

"""
In [62]: (executing lines 520 to 534 of "correcTP4-Fonctions-Suites-Flottants.py")
n : 0 Bibliothèque : 9.357623e-14 et     Maison : 1.000000e+00
n : 1 Bibliothèque : 9.357623e-14 et     Maison : -2.900000e+01
n : 2 Bibliothèque : 9.357623e-14 et     Maison : 4.210000e+02
.....................................o(
n : 21 Bibliothèque : 9.357623e-14 et     Maison : -1.194482e+11
n : 22 Bibliothèque : 9.357623e-14 et     Maison : 1.597426e+11
n : 23 Bibliothèque : 9.357623e-14 et     Maison : -2.044193e+11
n : 24 Bibliothèque : 9.357623e-14 et     Maison : 2.507830e+11
..................................
n : 72 Bibliothèque : 9.357623e-14 et     Maison : 1.074591e+02
n : 73 Bibliothèque : 9.357623e-14 et     Maison : -4.373428e+01
n : 74 Bibliothèque : 9.357623e-14 et     Maison : 1.756034e+01
..................................
n : 80 Bibliothèque : 9.357623e-14 et     Maison : 5.595934e-02
n : 81 Bibliothèque : 9.357623e-14 et     Maison : -2.053140e-02
n : 82 Bibliothèque : 9.357623e-14 et     Maison : 7.453018e-03
n : 83 Bibliothèque : 9.357623e-14 et     Maison : -2.661833e-03
..................................
n : 88 Bibliothèque : 9.357623e-14 et     Maison : 1.931170e-05
n : 89 Bibliothèque : 9.357623e-14 et     Maison : 1.687945e-06
..............................................
n : 92 Bibliothèque : 9.357623e-14 et     Maison : 6.257378e-06
n : 93 Bibliothèque : 9.357623e-14 et     Maison : 6.053660e-06
n : 94 Bibliothèque : 9.357623e-14 et     Maison : 6.118676e-06
n : 95 Bibliothèque : 9.357623e-14 et     Maison : 6.098145e-06
..................................
n : 102 Bibliothèque : 9.357623e-14 et     Maison : 6.103044e-06
n : 103 Bibliothèque : 9.357623e-14 et     Maison : 6.103042e-06
.............................
n : 118 Bibliothèque : 9.357623e-14 et     Maison : 6.103042e-06
n : 119 Bibliothèque : 9.357623e-14 et     Maison : 6.103042e-06
"""
```

5.4 Exercice 24

```
def somme1(n):
    '''Somme du plus grand au plus petit'''
    s = 0
    for i in range(1,n+1):
        s += 1/i**4
```

```
    return s

def somme2(n):
    '''Somme du plus petit au plus grand'''
    s = 0
    for i in range(n+1,0,-1):
        s += 1/i**4
    return s

for n in [10000, 100000]:
    print('Somme 1 = ', somme1(n))
    print('Somme 2 = ', somme2(n))

#In [69]: (executing lines 594 to 610 of "correcTP4-Fonctions-Suites-Flottants.py")
# Somme 1 =  1.0823232337108615
# Somme 2 =  1.082323233710805
# Somme 1 =  1.0823232337108615
# Somme 2 =  1.082323233711138

#valeur de la limite pi**4/90
# In [70]: math.pi**4/90
# Out[70]: 1.082323233711138
```

5.5 Exercice 25

```
def boulanger(x):
    '''Fonction du boulanger'''
    #0.5 est un flottant représenté de façon exacte
    if 0<= x <= 0.5:
        return 2*x
    return 2*(1 - x)

def iterations(x,n):
    '''tableaux des n premières itérations par la fonction du boulanger
    en partant de t'''
    tab = [x]
    for k in range(n):
        tab.append(boulanger(tab[-1]))
    return tab
```

Soit $b(n)$ le terme de rang n d'une suite du boulanger. Si $0 \leqslant b(n) \leqslant 1/2$ son développement binaire est du type $0, 0 \ldots$. $b(n+1) = 2b(n)$ donc son développement binaire est obtenu à partir de celui de $b(n)$ en décalant tous les bits vers la gauche. Si $1/2 \leqslant b(n) \leqslant 1$ son développement binaire est du type $0, 1 \ldots$. $b(n+1) = 2(1 - b(n))$ donc son développement binaire est obtenu à partir de celui de $b(n)$ en remplaçant tous les 0 par des 1 et vice-versa puis en décalant tous les bits vers la gauche. Pour la valeur initiale $b(0) = 1/3$ on a $b(1) = 2/3$ $b(2) = 2/3$ et par récurrence immédiate, pour tout entier $n \geqslant 1$ on a $b(n) = 2/3$ La représentation de $1/3$ sous forme de flottant n'est pas exacte, on peut l'obtenir avec la fonction `Decimal.from_float` du module decimal.

```
"""
In [34]: Decimal.from_float(1/3)
Out[34]: Decimal('0.333333333333333314829616256247390992939472198486328125')
"""
```

Il est donc normal qu'avec une représentation en flottants cette suite se comporte de façon chaotique. Si on calcule les

100 premiers termes, on observe que les derniers sont nuls au lieu d'etre égaux à 2/3. Le module `decimal` permet une représentation avec un nombre réglable de décimales exactes. Avec 50 décimales exactes, les termes de rang compris entre 0 et 100 sont plus proches de 2/3 que pour les flottants mais ils s'éloignent progressivement des valeurs exactes.

```
"""
In [16]: t1 = iterations(1/3, 100)

In [17]: t1[-5:]
Out[17]: [0.0, 0.0, 0.0, 0.0, 0.0]

In [18]: from decimal import *

In [19]: getcontext().prec = 50

In [20]: t2 = iterations(Decimal('1')/Decimal('3'), 100)

In [21]: t2[-4:]
Out[21]:
[Decimal('0.66666666666666666666693076054171421445864514650112'),
 Decimal('0.66666666666666666666613847891657157102709706699776'),
 Decimal('0.66666666666666666666772304216685685783458058600448'),
 Decimal('0.66666666666666666666455391566628628433083882799104')]

In [49]: t2[:4]
Out[49]:
[Decimal('0.33333333333333333333333333333333333333333333333333'),
 Decimal('0.66666666666666666666666666666666666666666666666666'),
 Decimal('0.66666666666666666666666666666666666666666666666668'),
 Decimal('0.66666666666666666666666666666666666666666666666664')]
"""
```

6 Pour ceux qui s'ennuient

6.1 Exo 27 Exponentiation rapide

```
def expo_rapideiter(a, n): #Version itérative
    acc =  a
    puis = 1
    while n > 0:
        if n%2 == 1:
            puis *= acc
        acc **= 2
        n //= 2
    return puis

def expo_rapiderec(a, n): #Version récursive
    if n == 0:
        return 1
    if n%2 == 0:
        return expo_rapiderec(a, n//2)**2
    return expo_rapiderec(a, n//2)*a
```

6.2 Exo 28 Projet Euler 56

```
def digits(n, base = 2):
    t = [n%base]
    n = n // base
    while n > 0:
        t.append(n%base)
        n = n // base
    t.reverse()
    return t

def projet56():
    maxi = -1
    for a in range(2, 101):
        for b in range(2, 101):
            s = sum(digits(expo_rapideiter(a,b), base = 10))
            if s > maxi: maxi = s
    return maxi

"""
In [60]: projet56()
Out[60]: 972
"""
```