```
# −*− coding: utf−8 −*−
"""
Created on Tue Jun 25 08:39:47 2013

@author: stephane

Corrigé du (long) TD d'initiation

Stage informatique à Luminy, avril 2014
"""

#
# EXO 1 : done
#

#
# EXO 2 : project euler numéro 1
#

s = 0

for i in range(1000):
    if i%3 != 0 and i%5 != 0:
        s = s + i  # Ou encore : s += i

print("La somme vaut %i" % s)

# La somme vaut 266332
#
# Complément : et si on remplace 10**3 par 10**10 ?


#
# EXO 3 : bissextile
#

def bissextile(n):
    """ L'année n est−elle bissextile ?"""
    return n%4 == 0 and ((n%100 != 0) or (n%400 == 0))


#
# EXO 4 : exponentiation
#

def expo_rec(x,n):
    """ Calcul récursif de x**n"""
    if n == 0:
        return 1
    return x * expo_rec(x,n−1)

def expo_it(x,n):
    """ Calcul itératif de x**n"""
    res = 1
    for _ in range(n):
        res = res * x
    return res

def expo_rap_rec(x,n):
    """ Calcul de x**n par exponentiation rapide récursive"""
    if n == 0:
        return 1
    if n %2 == 0:
        return expo_rap_rec(x,n/2)**2
    return x * expo_rap_rec(x,n/2)**2

def expo_rap_it(x,n):
    """ Calcul de x**n par exponentiation rapide itérative"""
    N, p, res = n, x, 1
```



```
    while N > 0:
        if N % 2 == 1:
            res = res * p   # Ou encore : res *= p
        p = p * p
        N = N // 2          # Ou N //= 2
    return res

# Exercice : comprendre ce qui précède :−)
# Comment économiser une multiplication ?

print(3**19, expo_rec(3,19), expo_it(3,19),
      expo_rap_rec(3,19), expo_rap_it(3,19))
# (1162261467, 1162261467, 1162261467, 1162261467, 1162261467)

#
# EXO 5 : factorielle
#

def facto_rec(n):
    """ Calcul récursif de n!"""
    if n == 0:
        return 1
    return n * facto_rec(n−1)

def facto_it(n):
    """ Calcul itératif de n!"""
    res = 1
    for i in range(1, n+1): # attention !
        res *= i
    return res

from math import factorial, sqrt

print(facto_rec(10), facto_it(10), factorial(10))
# (3628800, 3628800, 3628800)

#
# EXO 6 : Fibonacci
#

def fibo_rec(n):
    """ Fibonacci, version récursive naïve"""
    if n <= 1:
        return n
    return fibo_rec(n−1) + fibo_rec(n−2)

def fibo_it(n):
    """ Fibonacci, version itérative"""
    if n <= 1:
        return n
    a, b  = 0, 1
    for _ in range(2, n+1):
        a, b = b, a+b
    return b

print(fibo_rec(10), fibo_it(10))
# (55, 55)

# fibo_rec(100) fera trop d'appels récursifs −> no way !

#>>> fibo_it(100)
#354224848179261915075L

# programmons une espèce d'option remember à la maple :

fibo_memo = {0:0, 1:1}

def fibo_memoization(n):
    """ Fibonacci : avec memoïzation"""
```

```
    if n in fibo_memo:
        return fibo_memo[n]
    res = fibo_memoization(n−1) + fibo_memoization(n−2)
    fibo_memo[n] = res
    return res

#>>> fibo_memoization(100)
#354224848179261915075L

#
# EXO 7 : somme des décimales
#

def phi_rec(n):
    """ Calcul récursif de la somme des décimales"""
    if n < 10:
        return n
    return (n % 10) + phi_rec(n // 10)

def phi_it(n):
    """ Calcul itératif de la somme des décimales"""
    s, r = 0, n
    while r > 0:
        s += (r % 10)
        r //= 10
    return s

print(phi_rec(facto_it(100)), phi_it(facto_it(100)), phi_it(facto_it(500)))
#(648L, 648L, 4599L)
# exercice : expliquer les «L»

#
# EXO 8 : primalité
#

def premier(n):
    """ Test élémentaire de primalité"""
    if n <= 1:
        return False
    t = 2
    while t <= sqrt(n):
        if n % t == 0:
            return False
        t += 1
    return True

print [i for i in range(50) if premier(i)]
# [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

nb = 0
for i in range(1001):
    if premier(i):
        nb += 1

print(nb)
# 168

# On aurait pu écrire :
# print(len([i for i in range(1000) if premier(i)]))

# La complexité en en n^{3/2}/ln(n)
# ça passe encore pour n=10**6 mais pas pour n=10**7 !

#
# EXO 9 : algorithme d'Euclide
#

def pgcd_rec(a,b):
    """ pgcd de deux entiers positifs − version récursive"""
```



```
    if a < b:
        return pgcd_rec(b, a)
    if b == 0:
        return a
    return pgcd_rec(b, a%b)

def pgcd_iter(a,b):
    """ pgcd de deux entiers positifs − version itérative"""
    g, d = max(a,b), min(a,b)
    while d != 0:
        g, d = d, g%d
    return g

print(pgcd_iter(1515, 1789), pgcd_rec(1515, 1789))

def bezout_rec(a,b):
    """ calcul récursif des coefficients de Bezout"""
    if a < b:
       u, v = bezout_rec(b, a)
       return v, u
    if b == 0:
       return(1, 0)
    u, v = bezout_rec(b, a%b)
    return v, u−v*(a//b)

print(bezout_rec(1515, 1789))
# (−111, 94)

#>>> −111*1515+94*1789
#1

#
# EXO 10 : suite de Syracuse
#

def syracuse(n):
    """ Longueur de la suite de Syracuse issue d'un entier"""
    if n == 1 :
        return 0
    if n % 2 == 0:
        return 1 + syracuse(n//2)
    return 1 + syracuse(3*n+1)

print(map(syracuse,[1,2,3,10,15,127]))

# [0, 1, 7, 6, 17, 46]

#print(max(syracuse(i) for i in range(1,1001)),
#      max(syracuse(i) for i in range(1,1000001)))
# (178, 524)

# On peut faire des variantes, en calculant le plus haut sommet atteint...

#
# EXO 11 : quel jour sommes−nous ?
#


def premier_janvier(n):
    """ calcul du jour dans la semaine (lundi = 1)
    du premier janvier de l'annnée n"""
    if n == 2013:
        return 2
    j = 2
    if n > 2013:
        for annee in range(2013,n):
            if bissextile(annee):
                j += 2
            else:
```

```
                j += 1
        return ((j-1) % 7) + 1
    for annee in range(2012, n-1, -1):
        if bissextile(annee):
            j -= 2
        else:
            j -= 1
    return ((j-1) % 7) + 1

print(premier_janvier(2042), premier_janvier(1917))

#(3, 1)

def jour(date, mois, annee): #janvier=1
    """ Calcul du jour dans la semaine pour une date donnée"""
    j = premier_janvier(annee)
    durees = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30]
    if bissextile(annee):
        durees[1] += 1
    for m in range(mois-1):
        j += durees[m]
    j += date-1
    return ((j-1)%7)+1

print(jour(22, 4, 2014))
# 2

#
# EXO 12 : done
#

jouet1 = [1, 4, 12, -5, -3]
jouet2 = [i**2 for i in range(50)]

#>>> jouet2
#[0, 1, 4, 9, ... , 2209, 2304, 2401]

#
# EXO 13 : promis...
#

#
# EXO 14 : somme d'un tableau
#


def somme1(t):
    """ Calcul de la somme des éléments d'un tableau"""
    s = 0
    for i in range(len(t)):
        s += t[i]
    return s

# mieux :

def somme2(t):
    """ Calcul de la somme des éléments d'un tableau"""
    s = 0
    for x in t:
        s += x
    return s

# encore mieux :

def somme3(t):
    """ Calcul de la somme des éléments d'un tableau"""
    return sum(x for x in t)

# et finalement :
```



```
def somme4(t):
     """ Calcul de la somme des éléments d'un tableau"""
     return sum(t)


print(somme1(jouet2),somme2(jouet2),somme3(jouet2),somme4(jouet2))
#(40425, 40425, 40425, 40425)

#
# EXO 15 : maximum
#

# NB : il y a déjà une fonction max...

def maxi(t):
    """ Calcul du maximum des éléments d'un tableau"""
    assert len(t) > 0
    prov = t[0] # maximum provisoire
    for x in t: # Et non, ce ne serait pas une bonne idée de prendre t[1:]
        if x > prov:
            prov = x
    return prov

def pos_maxi(t):
    """ Calcul de la position du maximum des éléments d'un tableau"""
    assert len(t) > 0
    pos = 0
    for i in range(1,len(t)):
        if t[i] > t[pos]: # remplacer par >= si on veut la dernière position
            pos = i
    return pos

print(maxi(jouet1),pos_maxi(jouet1),maxi(jouet2),pos_maxi(jouet2))
#(12, 2, 2401, 49)

#
# EXO 16 : moyenne et variance
#

def moyenne(t):
    """ Calcul de la moyenne des éléments d'un tableau"""
    return float(somme1(t)) / len(t)  # float pour Python2

def variance1(t):
    """ Calcul de la variance des éléments d'un tableau"""
    s1, s2 = 0., 0.     # et pas 0, 0 : toujours pour Python2
    for x in t:
        s1 += x
        s2 += x**2
    return s2/len(t)-(s1/len(t))**2

def variance2(t):
    """ Calcul de la variance des éléments d'un tableau : malin !"""
    return moyenne(map(lambda x:x**2,t))-moyenne(t)**2

print(moyenne(jouet1), variance1(jouet1), variance2(jouet1))
#(1.8, 35.76, 35.76)
print(moyenne(jouet2), variance1(jouet2), variance2(jouet2))
#(808.5, 534661.05, 534661.05)

#
# EXO 17 : tableau positif
#

def positif(t):
    """ t est-il constitué d'entiers/floats positifs ?"""
    for x in t:
        if x < 0:
```

```
            return False
    return True

print(positif(jouet1), positif(jouet2))

#(False, True)
# Ma première version était fausse : «return True» mal indenté...
# Comme quoi, faire des tests...

#
# EXO 18 : tableau croissant
#

def croissant(t):
    """ t est−il croissant ?"""
    for i in range(len(t)−1):
        if t[i+1] < t[i]:
            return False
    return True

print(croissant(jouet1),croissant(jouet2))
#(False, True)

#
# EXO 19 : monoreturn !
#

def positif2(t):
    """ t est−il constitué d'entiers/floats positifs ? monoreturn"""
    itsallgood = True
    indice = 0
    while itsallgood and indice < len(t):
        itsallgood = t[indice]>=0
        indice += 1
    return itsallgood

print(positif2(jouet1), positif2(jouet2))
#(False, True)

def croissant2(t):
    """ t est−il croissant ? monoreturn"""
    itsallgood = True
    indice = 0
    while itsallgood and indice < len(t)−1:
        itsallgood =  t[indice] <= t[indice+1]
        indice += 1
    return itsallgood

print(croissant2(jouet1), croissant2(jouet2))
#(False, True)


#
# EXO 20 : inversions
#

def inversions(t):
    """ Calcul du nombre d'inversions dans un tableau"""
    nb = 0
    for i in range(len(t)−1):
        for j in range(i+1, len(t)):
            if t[i] > t[j]:
                nb += 1
    return nb

print(inversions(jouet1), inversions(jouet2))

#
# EXO 21 : appartenance à un tableau
```



```
#

def appartient_a(x, t):
    """ Est−ce que x appartient à t ?"""
    for y in t:
        if x == y:
            return True
    return False

print(appartient_a(2304, jouet2), appartient_a(2305, jouet2))
#(True, False)
# En fait, «x in t» fait l'affaire !

#
# EXO 22 : premiere position
#

def premiere_position(x, t):
    """ Première position de x dans t"""
    for i in range(len(t)):
        if t[i] == x:
            return i
    return None

# NB : la méthode index fait ça...

jouet3 = [1, 5, 21, 42, 5, 1]

print(premiere_position(5, jouet3), jouet3.index(5))
# (1, 1)
#print(premiere_position(15, jouet3), jouet3.index(15))
# ValueError: 15 is not in list

def derniere_position1(x, t):
    """ Dernière position de x dans t"""
    p = None
    for i in range(len(t)):
        if t[i] == x:
            p = i
    return p

def derniere_position2(x, t):
    """ Dernière position de x dans t"""
    for i in range(len(t)−1, −1, −1):
        if t[i] == x:
            return i
    # Le None est retourné par défaut


print(derniere_position1(5, jouet3), derniere_position2(5, jouet3))
#(4,4)
print(derniere_position1(15, jouet3), derniere_position2(15, jouet3))
#(None, None)

#
# EXO 23 : positions
#

def positions(x, t):
    """ Liste des positions de x dans t"""
    pos = []
    for i in range(len(t)):
        if t[i] == x:
            pos.append(i)
    return pos

print(positions(15, jouet3), positions(5, jouet3))
#([], [1, 4])
```

```
#
# EXO 24 : recherche dichotomique
#

def recherche_dicho(x, l):
    """ Recherche dichotomique de x dans t supposé croissante"""
    debut, fin = 0, len(l)−1
    while fin >= debut:  # À la sortie, il n'y a plus rien !
        milieu = (debut + fin) // 2
        if l[milieu] == x:
            return True
        if l[milieu] > x:
            fin = milieu−1
        else:
            debut = milieu+1
    return False

print(recherche_dicho(960,jouet2),recherche_dicho(961,jouet2),
      recherche_dicho(4,jouet1), recherche_dicho(−5,jouet1))
#(False, True, True, False)
# Sans la croissance, la correction n'est pas assurée !

#
# EXO 25 : recherche d'un sous−mot
#

def sous_mot1(m, w):
    """ Retourne la position éventuelle de la première occurence
    de m dans w; None sinon"""
    for p in range(len(w)−len(m)):
        if w[p : p+len(m)] == m:
            return p
    return None

#>>> sous_mot1('titi','tructititoto')
#4
#>>> sous_mot1('tagada','tructititoto')

#
# EXO 26 : recherche d'un sous−mot, sans slicing
#

def sous_mot2(m,w):
    """ Retourne la position éventuelle de la première occurence
    de m dans w; None sinon. Pas de slicing !"""
    for p in range(len(w)−len(m)):
        ok = True # Parions qu'on va trouver m en position p
        i = p # le prochain caractère à vérifier
        while ok and i < p+len(m):
            if w[i] != m[i−p]:
                ok = False
            i += 1
        if ok:
            return p
    return None

#>>> sous_mot2('titi','tructititoto')
#4
#>>> sous_mot2('tagada','tructititoto')

#
# EXO 27 : boîte à outils
#

def dimensions(m):
    """ Dimensions d'une matrice"""
    return len(m), len(m[0]) # lignes, colonnes

#>>> m
```



```
#[[0, 0, 0], [0, 0, 0]]
#>>> dimensions(m)
#(2, 3)

def matrice_nulle(n, p):
    """ Retourne une matrice nulle de dimensions (n,p) """
    return [[0]*p for i in range(n)]

#>>> m = matrice_nulle(3,2)
#>>> m
#[[0, 0], [0, 0], [0, 0]]

def matrice_identite(n):
    """ Retourne une matrice identité (n,n) """
    m = matrice_nulle(n,n)
    for i in range(n):
        m[i][i] = 1
    return m

#>>> matrice_identite(4)
#[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]


#
# EXO 28 : matrice générique
#

def generique(n, p, f):
    """ Retourne une matrice (n,p) avec f(i+1,j+1) en position (i,j) """
    return [[f(i+1, j+1) for j in range(p)] for i in range(n)]

#>>> def f0(i, j): return abs(i−j)
#...
#>>> generique(3, 3, f0)
#[[0, 1, 2], [1, 0, 1], [2, 1, 0]]
#>>> generique(3, 3, lambda i, j:0)
#[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
#>>> def f(i, j):
#        if i ==j : return 1
#        return 0
#>>> generique(3, 3, f)
#[[1, 0, 0], [0, 1, 0], [0, 0, 1]]

#
# EXO 29 : matrice symétrique ?
#

def est_symetrique(M):
    """ Teste le caractère symétrique de M"""
    n, p = dimensions(M)
    if n <> p:
        return False
    for i in range(1, n):
        for j in range(i):
            if M[i][j] != M[j][i]:
                return False
    return True

#
# EXO 30 : transposition, somme, produit
#


def transposee(M):
    """ Calcule la transposée de M"""
    n, p = dimensions(M)
    N = matrice_nulle(p, n)
    for i in range(n):
        for j in range(p):
```

```
            N[j][i] = M[i][j]
    return N

# On aurait pu faire du [[... for i in ...] for j in ...]
# Mais
# 1) ça fait mal à la tête
# 2) c'est risqué
# 3) ça n'impressione pas grand monde.

def somme(M, N):
    """ Calcule la somme de deux matrices"""
    n1, p1 = dimensions(M)
    n2, p2 = dimensions(N)
    assert n1 == n2 and p1 == p2
    P = matrice_nulle(n1,p1)
    for i in range(n1):
        for j in range(p1):
            P[i][j] = M[i][j] + N[i][j]
    return P

def produit(M,N):
    """ Calcule le produit de deux matrices"""
    n1, p1 = dimensions(M)
    n2, p2 = dimensions(N)
    assert n2 == p1
    P = matrice_nulle(n1,p2)
    for i in range(n1):
        for j in range(p2):
            for k in range(p1):
                P[i][j] += M[i][k] * N[k][j]
    return P

#>>> m = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
#>>> transposee(m)
#[[1, 2, 3], [0, 1, 2], [1, 0, 1]]
#>>> somme(m,m), produit(m,m)
#([[2, 0, 2], [4, 2, 0], [6, 4, 2]], [[4, 2, 2], [4, 1, 2], [10, 4, 4]])

#
# EXO 31 : du pivot
#

def chercher_pivot(A, i):
    """ Donne la ligne du pivot de module maximale sur la i−ème colonne
    à partir de la i−ème ligne"""
    n, _ = dimensions(A) # le nombre de lignes
    j = i # la ligne du maximum provisoire
    for k in range(i+1, n):
        if abs(A[k][i]) > abs(A[j][i]):
            j = k # un nouveau maximum
    return j # en faisant bien attention à l'indentation :-)

def transvection_ligne(A, i, j, mu):
    """ L_i <− L_i + mu.L_j """
    _, nc = dimensions(A) # le nombre de colonnes
    for k in range(nc):
        A[i][k] += (mu * A[j][k])

def echange_lignes(A, i, j):
    """ Guess what..."""
    nc = len(A[0])
    for k in range(nc):
        A[i][k], A[j][k] = A[j][k], A[i][k]

def copie_matrice(M):
    """ Copie d'un tableau bidimensionnel"""
    n, _ = dimensions(M)
    R = [None] * n
    for i in range(n):
```



```
        R[i] = M[i][:]
    return R

def dilatation_ligne(A, i, mu):
    """ L_i <− mu * L_i"""
    _, nc = dimensions(A) # le nombre de colonnes
    for k in range(nc):
        A[i][k] *= mu

def inversion(A0):
    """Calcul de B0= A0**(−1) """
    A = copie_matrice(A0)
    n, p = dimensions(A)
    assert n == p
    B = matrice_identite(n)
    for i in range(n):
        j = chercher_pivot(A, i)
        if j > i:
            echange_lignes(A, i, j)
            echange_lignes(B, i, j)
        for k in range(i+1, n):
            x = A[k][i]/float(A[i][i]) # Pour Python 2
            transvection_ligne(A, k, i, −x)
            transvection_ligne(B, k, i, −x)
    for i in range(n−1,−1,−1):
        dilatation_ligne(B, i, 1./A[i][i])
        dilatation_ligne(A, i, 1./A[i][i])
        for j in range(i+1,n):
            transvection_ligne(B, i, j, −A[i][j])
    return B

#>>> inversion([[1,2],[3,4]])
#[[−1.9999999999999996, 0.9999999999999998], [1.4999999999999998, −0.4999999999999999]]

#
# EXO 32 : crible d'Ératosthène
#

def crible(n):
    """ Retourne la liste des nombres premiers majorés par n"""
    t = [True] * (n+1)
    p = 2
    while p**2 <= n:
        if t[p]:
            for i in range(2, n/p +1):
                t[i*p] = False
        p += 1
#    return [i in range(n+1) if t[i]] # C'est la version idiomatique
    liste = []
    for i in range(2, n+1):
        if t[i]:
            liste.append(i)
    return liste

#>>> crible(50)
#[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
# Complexité : n*ln(n)

#>>> len(crible(10**6)), len(crible(10**7))
#(78498, 664579)

#
# EXO 33 : permutations
#

def verifier_permutation_feignasse(t):
    """ le tableau t de longueur n est−il une permutation de {0,1,...,n−1} ?"""
    n = len(t)
```

```
    return set(t) == set(range(n)) # bouhahaha !

def verifier_permutation(t):
    """ le tableau t de longueur n est−il une permutation de {0,1,...,n−1} ?"""
    n = len(t)
    vues = [False] * n # les valeurs déjà vues
    for x in t:
        if x < 0 or x >= n or vues[x]:
            return False
        vues[x] = True
    return True

#>>> verifier_permutation([1,2,3,0]), verifier_permutation([1,2,3,1]), verifier
#_permutation([1,2,3,1])
#(True, False, False)

def inverse(t):
    inv = [0] * len(t)
    for i in range(len(t)):
        inv[t[i]] = i
    return inv

#>>> inverse([1, 2, 3, 0, 6, 5, 4])
#[3, 0, 1, 2, 6, 5, 4]

def compose(p1, p2):
    return [p1[p2[i]] for i in range(len(p1))]

#>>> x = [1,2,3,0,6,5,4]
#>>> compose(x,x)
#[2, 3, 0, 1, 4, 5, 6]

#
# EXO 34 : un tri
#

def tri_selection(t): # tri en place; ne retourne rien
    n = len(t)
    for k in range(n−1, 0, −1): # k : n−1−>1
        pos_mp = 0 # la position du maximum provisoire
        for i in range(1, k+1):
            if t[i] > t[pos_mp]: # on a un meilleur candidat
                pos_mp=i
        t[pos_mp], t[k] = t[k], t[pos_mp] # échange des deux éléments
```