Correction du concours blanc

Informatique pour tous, première année

Julien Reichert

```
### C-1

def variance(L):
    total, somme, sommecarres = 0, 0, 0
    for element in L:
        total, somme, sommecarres = total + 1, somme + element, sommecarres + element ** 2
    return (sommecarres - somme ** 2) / total # Koenig-Huygens

# Complexité linéaire (on évitera de calculer la moyenne dans chaque tour de boucle).

### C-2

def nb_occ(x, L):
    rep = 0
    for elt in L:
        rep += elt == x # petite astuce amusante
    return rep

### E-1

def nbinter(L, LL):
    intersections = 0
    for elt in L:
        for eltelt in LL:
            intersections += elt == eltelt
    return intersections

# Complexité quadratique, en l'occurrence le produit des tailles des listes.
# On peut grandement accélérer via des tables de hachage avec la fonction ci-dessous.

def nbintercheat(L, LL):
    Ld, LLd = {}, {}
    for elt in L:
        try: Ld[elt] += 1
        except: Ld[elt] = 1
    for elt in LL:
        try: LLd[elt] += 1
        except: LLd[elt] = 1
    intersections = 0
    for cle in Ld:
        try: intersections += Ld[cle] * LLd[cle]
        except: pass
    return intersections

# 0,63 s pour deux listes de taille un million avec des nombres aléatoires entre 1 et 100000.
# La fonction nbinter a été arrêtée après plus d'une heure de moulinage sur les mêmes entrées.
```

```
### E-2

import math

def moyag(a, b, epsilon):
    assert a >= 0 and b >= 0 and epsilon >= 0 # peu importe l'ordre en pratique
    while abs(b - a) > 2*epsilon: # économie de bout de chandelle
        a, b = (a+b)/2, math.sqrt(a*b) # on peut modifier les arguments numériques sans problème
    return (a+b)/2 # forcément à epsilon de la limite

# On pose comme variant la partie entière du logarithme en base 2 de (b-a)/epsilon.
# Cette valeur, entière, baisse d'au moins un à chaque tour
# car a devient la moyenne arithmétique de a et b, donc l'écart entre b et a diminue au moins de moitié.
# Ceci prouve la terminaison car si le variant devient négatif on sort de la boucle.

### Question 1

def integrale_rectangles(f, a, b, h):
    assert a <= b and h >= 0
    x, somme, Df = a, 0, (f(b)-f(a)) / (b-a)
    while x+h < b: # ce sera plus simple
        somme += f(x+h)*h
        fp = (f(x+h) - f(x)) / h
        x += h
        if fp > Df:
            h *= .9
        if fp < Df:
            h *= 1.1
    return somme + f(b)*(b-x) # la preuve

# Après un test sommaire, c'est moins précis que la méthode habituelle.

### Question 2

import math

def euler_richardson(F, a, b, y0, h, epsilon):
    X = [a]
    Y = [y0]
    while X[-1] + h < b:
        yp = F(Y[-1], X[-1])
        yp2 = F(Y[-1] + yp * h/2, X[-1] + h/2)
        eta = h/2 * abs(yp2-yp)
        alpha = eta / epsilon
        if alpha < 1:
            X.append(X[-1] + h)
            Y.append(Y[-1] + h * yp2)
        h = .9 * h / math.sqrt(alpha)
    X.append(b)
    Y.append(Y[-1] + F(Y[-1], X[-2]) * (b-X[-2]))
    return X, Y

# Test pour l'équation différentielle de solution la fonction exponentielle (entre 0 et 3) :
# C'est beaucoup plus précis que la méthode de base avec le pas initial proposé (un cent-millième),
# ainsi que la méthode de base avec les mêmes abscisses (au nombre de 12234 seulement, d'ailleurs).
```