Correction du concours blanc

Informatique pour tous, première année

Julien Reichert

```
### C-1

def moyenne(l):
    s = 0
    for i, x in enumerate(l):
        s += x
    return s / (i+1) # petite astuce amusante

### C-2

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

### PO

def all_diff(l):
    if l == []:
        return True
    ll = sorted(l)
    x = ll[0]
    for i in range(1, len(ll)):
        if x == ll[i]:
            return False
        x = ll[i]
    return True

def all_diff_une_ligne(l):
    return max([nb_occ(x, l) for x in l]) == 1

def all_diff_standard(l):
    for i in range(len(l)):
        if l[i] in l[i+1:]:
            return False
    return True

### C-4

def min_acceptable(lind, l):
    mini = None
    for ind in lind:
        if 0 <= ind < len(l) and (mini == None or mini > l[ind]):
            mini = l[ind]
    return mini
```

C-5

Les deux premières linéaires en la taille de la liste,
la troisième quadratique en sa taille (sauf avec un tri, mais il faut attendre la SPE pour le savoir),
la quatrième linéaire en la taille de lind (mais mal écrit, cela serait le produit des deux tailles).

Question 1

```
def evalue_poly(l,x):
    xpuissancen = 1
    somme = 0
    for i in range(len(l)):
        somme += l[i] * xpuissancen # plus efficace que l[i] * x ** i
        xpuissancen *= x
    return somme
```

Question 2

```
def integre_poly(l):
    ll = [0]
    for i in range(len(l)):
        ll.append(l[i]/(i+1))
    return ll
```

Question 3

```
def interpole(X, Y):
    coeff2 = Y[2] / ((X[2]-X[0])*(X[2]-X[1]))
    coeff1 = Y[1] / ((X[1]-X[0])*(X[1]-X[2]))
    coeff0 = Y[0] / ((X[0]-X[2])*(X[0]-X[1]))
    p2 = coeff2 + coeff1 + coeff0
    p1 = - (X[0]+X[1]) * coeff2 - (X[0]+X[2]) * coeff1 - (X[1]+X[2]) * coeff0
    p0 = X[0]*X[1] * coeff2 + X[0]*X[2] * coeff1 + X[1]*X[2] * coeff0
    return [p2, p1, p0]
```

Question 4

```
def integrale_rectangles(f, bornes):
    somme = 0
    for i in range (len(bornes)-1):
        deb, fin = bornes[i], bornes[i+1]
        somme += (fin-deb) * f((deb+fin)/2) # méthode au milieu
    return somme
```

Question 5

```
def simpson(f, bornes):
    somme = 0
    for i in range (len(bornes)-1):
        deb, fin = bornes[i], bornes[i+1]
        mil = (deb+fin)/2
        poly = interpole((deb, mil, fin), (f(deb), f(mil), f(fin)))
        prim = integre_poly(poly)
        somme += evalue_poly(prim, fin) - evalue_poly(prim, deb)
    return somme
```