Correction du DS 2

Julien Reichert

Partie 1

```
# Exercice 1.1

def nb_debut_voy(l):
    d = dict()
    for mot in l:
        if mot[0] in "aeiouy":
            d[mot] = 1
    return len(d)

# Exercice 1.2

def liste_debut_voy(l):
    d = dict()
    for mot in l:
        if mot[0] in "aeiouy":
            d[mot[0]] = 1
    return list(d.keys())

# Exercice 1.3

def liste_lettres_mot(l):
    d = dict()
    for mot in l:
        for lettre in mot:
            d[lettre] = 1
    return list(d.keys())

# Exercice 1.4

def lettre_majoritaire(l):
    d = dict()
    maxi, maxlettre = 0, ''
    for mot in l:
        for lettre in mot:
            if lettre not in d:
                d[lettre] = 0
            d[lettre] += 1
            if d[lettre] > maxi:
                maxi, maxlettre = d[lettre], lettre
    return maxlettre
```

```
# Exercice 1.5

def taille_romain(n):
    assert 0 < n < 4000
    tailles = [0, 1, 2, 3, 2, 1, 2, 3, 4, 2]
    rep = 0
    for chiffre in str(n): # On passe par les chaînes pour faciliter les choses.
        rep += tailles[int(chiffre)]
    return rep

# Exercice 1.6

def plus_grand_carre_ou_cube(n): # Solution en temps constant, une boucle est aussi possible.
    carre = int(n ** 0.5) ** 2
    cube = int(n ** (1/3)) ** 3
    return max(carre, cube)

# Exercice 1.7

def phases_monotonie(l):
    if len(l) < 2:
        return len(l) # 0 pour 0, 1 pour 1
    i = 0
    while i < len(l)-1 and l[i] == l[i+1]: # On zappe le début constant.
        i += 1
    cro = (l[i] < l[i+1]) # Phase croissante ou non ?
    phases = 1
    for j in range(i, len(l)-1):
        if l[j] == l[j+1]: # Pas de changement.
            continue
        if (l[j] < l[j+1]) != cro: # Attention à parenthéser !
            cro = not cro # Inversion de la croissance.
            phases += 1 # Et nouvelle phase.
    return phases
```

Partie 2

Exercice 2.1

```
SELECT grand_prix, date FROM SAISONS WHERE continent = 'Europe'
```

Exercice 2.2

La requête calcule le nombre de points de chaque pilote (également précisé dans le résultat), a priori sans ordre particulier, sur l'ensemble des Grands Prix de la saison en cours (d'après l'énoncé).

Un résultat possible sera par exemple `Hamilton 363`.

Exercice 2.3

```
SELECT date, SAISONS.grand_prix FROM SAISONS JOIN GP ON SAISONS.grand_prix = GP.grand_prix
WHERE continent = 'Asie' AND cl_qualif = 1 AND cl_course = 1
AND date >= '2017-01-01' AND date <= '2017-12-31'
```

Exercice 2.4

```
SELECT pilote FROM GP GROUP BY pilote ORDER BY SUM(points) DESC LIMIT 1
```

Exercice 2.5

```
SELECT pilote FROM TOURS WHERE date='1994-05-01'
GROUP BY pilote
HAVING MAX(num_tour) = (SELECT MAX(num_tour) FROM TOURS WHERE date='1994-05-01')
ORDER BY SUM(temps_tour) LIMIT 1
```

Partie 3

Exercice 3.1

```
def somme_liste(l):
    rep = 0
    for x in l:
        rep += x
    return rep
```

Exercice 3.2

```
def somme_sous_liste(seq, indseq):
    rep = 0
    for ind in indseq:
        rep += seq[ind]
    return rep
```

Exercice 3.3

Telle qu'elle est écrite, la fonction est de complexité linéaire en la taille de `indseq` quelle que soit la taille de `seq`. Cela se voit immédiatement sur la structure de la fonction, avec un corps de boucle de complexité constante, des opérations en temps constant à l'extérieur de la boucle et le nombre de tours de boucle évident.

Exercice 3.4

Le nombre de sous-séquences d'une séquence de taille n étant 2^n, tester toutes les sous-séquences possibles jusqu'à trouver peut entraîner une complexité en $\mathcal{O}(n \times 2^n)$, ce qui est au-delà du raisonnable.

Exercice 3.5

```
def sommes_moins_plus(l):
    reps = [0, 0]
    for x in l:
        if x < 0:
            reps[0] += x
        else:
            reps[1] += x
    return reps
```

Exercice 3.6

Pour faire bonne mesure, on renvoie même une façon de produire la somme s'il en existe une et on déclenche une erreur sinon.

```
def subsetsum(l, s):
    d = { 0 : [] } # Initialisation nécessaire !
    for x in l:
        for cle in d:
            if cle+x not in d:
                d[cle+x] = d[cle] + [x]
            if cle+x == s:
                return d[cle+x] # Version de base : return True
    raise ValueError("Impossible de produire la somme") # Version de base : return False
```

Exercice 3.7

La complexité reste en pratique exponentielle dans le pire des cas, en fonction des valeurs dans la liste (typiquement s'il s'agit de puissances de 2), mais on a une majoration potentiellement plus raffinée : seules les sommes possibles seront considérées, et ce au plus une fois par élément de la liste. Donc en posant `[mini, maxi]` le résultat de la fonction écrite dans l'exercice 3.5, on a aussi une formule de majoration en $\mathcal{O}((max - min)n)$ où n est la taille de la liste.