Corrigé du contrôle d'informatique

Exercice 1

a) Il s'agit de vérifier que la suite représentée par la liste `lst` est bien strictement croissante et que son premier terme est strictement supérieur à 1 :

```
def estEgyptienne(lst):
    if not (lst[0] > 1):
        return False
    for i in range(len(lst)-1):
        if not (lst[i] < lst[i+1]):
            return False
    return True
```

Prendre garde à la valeur de l'intervalle à parcourir : si `lst` est égale à la liste $a_0, a_1, \dots, a_{n-1}$ l'inégalité $a_i < a_{i+1}$ doit être vérifiée pour tout i dans l'intervalle $[\![0, n-1[\![$.

b) Pour calculer la somme des fractions qui interviennent dans le calcul, on utilise la formule : $\dfrac{p}{q} + \dfrac{1}{m} = \dfrac{mp+q}{mq}$.

```
def rationnel(lst):
    p, q = 0, 1
    for m in lst:
        p, q = m * p + q, m * q
    return p, q
```

c) Représentons sous la forme d'un tableau les valeurs successives prises par les variables a, b et m au sein de la boucle conditionnelle :

k	1	2	3	4
a	19	18	14	6
b	20	40	120	1080
m	–	2	3	9

À l'entrée de la quatrième itération, $a = 6$, $b = 1080$ et `lst=[2, 3, 9]`. Mais cette fois a divise b donc $b/a = 180$ est ajouté à `lst` et le résultat renvoyé est la liste `[2, 3, 9, 180]`.

On peut observer que $\dfrac{19}{20} = \dfrac{1}{2} + \dfrac{1}{3} + \dfrac{1}{9} + \dfrac{1}{180}$.

d) Les valeurs m_k, a_{k+1} et b_{k+1} sont définies lorsque a_k ne divise pas b_k, et dans ce cas on a $m_k = \left\lfloor \dfrac{b_k}{a_k} \right\rfloor + 1$, $a_{k+1} = a_k m_k - b_k$ et $b_{k+1} = b_k m_k$.

e) Par définition de la partie entière, $m_k - 1 \leqslant \dfrac{b_k}{a_k} < m_k$. De plus, puisque a_k ne divise pas b_k, la première inégalité est stricte : $m_k - 1 < \dfrac{b_k}{a_k} < m_k$. On a $\dfrac{b_k}{a_k} < m_k$ donc $a_{k+1} = a_k m_k - b_k > 0$ ou encore, s'agissant d'un entier, $a_{k+1} \geqslant 1$. De plus, $m_k - 1 < \dfrac{b_k}{a_k}$ donc $a_{k+1} = a_k m_k - b_k < a_k$.

Tant qu'elle est définie, la suite (a_k) est une suite d'entier minorée et strictement décroissante. Elle ne peut donc décroitre indéfiniment et l'algorithme se termine.

f) On calcule $\dfrac{a_k}{b_k} - \dfrac{a_{k+1}}{b_{k+1}} = \dfrac{a_k}{b_k} - \dfrac{a_k m_k - b_k}{b_k m_k} = \dfrac{1}{m_k}$.

Notons $\mathrm{L} = [m_0, \dots, m_r]$ la liste renvoyée par cette fonction. Si $k < r$ on a $\dfrac{1}{m_k} = \dfrac{a_k}{b_k} - \dfrac{a_{k+1}}{b_{k+1}}$ et $\dfrac{1}{m_r} = \dfrac{a_r}{b_r}$ donc

$$\sum_{k=0}^{r} \frac{1}{m_k} = \sum_{k=0}^{r-1} \Big(\frac{a_k}{b_k} - \frac{a_{k+1}}{b_{k+1}} \Big) + \frac{a_r}{b_r} = \frac{a_0}{b_0} = \frac{p}{q}.$$

Il reste à vérifier que la représentation est bien conforme. Puisque $p < q$ on a $m_0 = \left\lfloor \frac{q}{p} \right\rfloor + 1 \geqslant 2$ et :

$$m_{k+1} > \frac{b_{k+1}}{a_{k+1}} = \frac{1}{a_k/b_k - 1/m_k} > \frac{1}{1/(m_k-1) - 1/m_k} = m_k(m_k-1).$$

Les deux inégalités $m_0 \geqslant 2$ et $m_{k+1} > m_k(m_k-1)$ permettent ensuite de prouver par récurrence que $2 \leqslant m_0 < m_1 < \cdots < m_r$.

g) Avec les notations de la question précédente, $1 \leqslant a_r < \cdots < a_1 < a_0 = p$ et s'agissant d'entiers, on a $r \leqslant p-1$. La liste renvoyée est donc au plus de longueur p.

Lorsque $p = n$ et $q = n! + 1$, on montre par récurrence sur $k \in [\![0, n-1]\!]$ que a_k et b_k sont définis et qu'il existe une constante entière α_k telle que $a_k = n-k$ et $b_k = \alpha_k(n-k)! + 1$.

- C'est vrai si $k = 0$, avec $\alpha_0 = 1$.
- Si $0 \leqslant k < n-1$, on suppose le résultat acquis au rang k. On a $n-k \geqslant 2$ donc a_k ne divise pas b_k. Les entiers m_k, a_{k+1} et b_{k+1} sont donc définis, et :

$$m_k = \alpha_k(n-k-1)! + 1, \quad a_{k+1} = m_k a_k - b_k = n-k-1, \quad b_{k+1} = m_k b_k = \underbrace{\left(\alpha_k^2(n-k)! + \alpha_k + \alpha_k(n-k)\right)}_{\alpha_{k+1}}(n-k-1)! + 1$$

  ce qui prouve le résultat au rang $k+1$.

La suite (a_k) est donc définie jusqu'au rang $n-1$ avec $a_{n-1} = 1$, ce qui assure la terminaison à ce rang. La fraction égyptienne obtenue est donc de longueur n.

Exercice 2

a) Une solution élégante consiste à utiliser la fonction `zip` qui énumère deux itérables en parallèle et qui stoppe dès que le plus petit des deux est parcouru dans son entier. À droite figure une version plus traditionnelle.

```python
def estPrefixe(u, v):
    if len(v) > len(u):
        return False
    for x, y in zip(u, v):
        if x != y:
            return False
    return True
```

```python
def estPrefixe(u, v):
    if len(v) > len(u):
        return False
    for i in range(len(v)):
        if u[i] != v[i]:
            return False
    return True
```

Examinons le cas où $u = aaa\cdots aaa$ (p fois la lettre a) et $v = aaa\cdots ab$ ($q-1$ fois la lettre a suivis de la lettre b). Notons $f(p,q)$ le nombre de comparaisons effectués entre caractères individuels.

Si $p < q$ on a $f(p,q) = 0$. Si $q \leqslant p$ on a $f(p,q) = q$, et cette quantité est bien évidemment maximale.

b) On a bord$(abaababa) = aba$ et bord$(abababa) = ababa$ (comme on peut le constater sur ce dernier exemple, les bords gauche et droite peuvent se chevaucher).

On peut utiliser la fonction précédente pour chercher parmi tous les suffixes de u ceux qui en sont aussi des préfixes ; si on commence la recherche à partir du plus grand des suffixes propres, le premier trouvé sera aussi le plus grand.

```python
def bord(u):
    for i in range(1, len(u)):
        v = u[i:]
        if estPrefixe(u, v):
            return v
    return ''
```

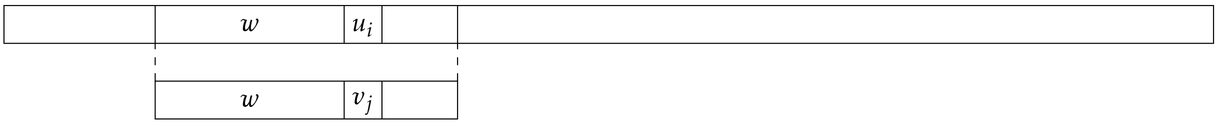
c) v est un facteur de u si et seulement si v est un préfixe d'un suffixe de u. D'où la fonction :

```python
def estFacteur(u, v):
    if len(v) > len(u):
        return False
    for i in range(len(u)-len(v)):
        if estPrefixe(u[i:], v):
            return True
    return False
```

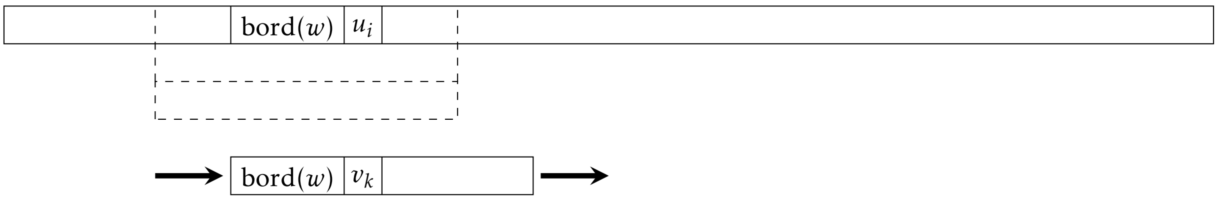
Considérons de nouveau le cas où $u = aaa\cdots aaa$ (p fois la lettre a) et $v = aaa\cdots ab$ ($q-1$ fois la lettre a suivis de la lettre b). Notons $g(p,q)$ le nombre de comparaisons effectués entre caractères individuels.

Si $p < q$ on a $g(p,q) = 0$. Si $p \geqslant q$ la fonction `est_prefixe` est utilisée $p-q+1$ fois, et à chaque fois q comparaisons sont effectuées (soit le nombre maximal de comparaisons). Le nombre total de comparaisons entre caractères individuels est donc égal à $g(p,q) = q(p-q+1)$.

d) Nous allons utiliser deux curseurs : l'indice i désigne un caractère du mot u et j un caractère du mot v. Considérons le moment où u_i est comparé à v_j :



- Si $u_i = v_j$ et v_j est la dernière lettre de v, on peut conclure : v est facteur de u ;
- si $u_i = v_j$ et v_j n'est pas la dernière lettre de v, l'algorithme se poursuit en comparant u_{i+1} et v_{j+1} ;
- si $u_i \neq v_j$ et $j = 0$, le mot v est décalé d'un cran et l'algorithme se poursuit en comparant u_{i+1} et v_0 ;
- enfin, si $u_i \neq v_j$ et $j > 0$, le mot v est décalé en fonction de la longueur du bord de w, et l'algorithme se poursuit en comparant u_i à v_k avec $k = |\mathrm{bord}(w)|$:




```python
def estFacteurMP(u, v, bord):
    i, j = 0, 0
    while len(v) - j <= len(u) - i:
        if u[i] == v[j]:
            if j == len(v) - 1:
                return True
            i, j = i + 1, j + 1
        else:
            if j == 0:
                i += 1
            else:
                j = bord[j]
    return False
```

Remarque. `len(v) - j` est le nombre de caractères dans v au delà de l'indice j, `len(u) - i` le nombre de caractères dans u au delà de l'indice i. Quand la première quantité dépasse la seconde, v ne peut plus être un facteur de u.