Corrigé du contrôle d'informatique

Exercice 1

Question 1. Il s'agit d'itérer n fois la suite (f_n, f_{n+1}) :

```
def f(n):
    x, y = 0, 1
    for k in range(n):
        x, y = y, x + y
    return x
```

On justifie la validité de cet algorithme par l'invariant : « *à l'entrée de la boucle d'indice k,* $x = f_k$ *et* $y = f_{k+1}$. »

Question 2. On procède de la même façon en utilisant l'invariant : « *à l'entrée de la boucle d'indice k,* $x = g_k$, $y = g_{k+1}$ *et* $z = g_{k+2}$. »

```
def g(n):
    x, y, z = 0, 1, 1
    for k in range(n):
        x, y, z = y, z, x + z
    return x
```

Question 3. Pour cette première version, on utilise un tableau qui mémorise toutes les valeurs précédentes de la suite h.

```
def h1(m, n):
    t = [None] * (n+1)
    t[0] = 0
    for k in range(1, m):
        t[k] = 1
    for k in range(m, n+1):
        t[k] = t[k-m] + t[k-1]
    return t[n]
```

Question 4. Mais on peut remarquer qu'il suffit de mémoriser les m dernières valeurs de la suite h. Dans ce cas, on utilise l'invariant : « *à l'entrée de la boucle d'indice* $k \geqslant m$ *on a* $t[(k-i) \bmod m] = h_{k-i}$ *pour* $1 \leqslant i \leqslant m$. »

```
def h2(m, n):
    t = [None] * m
    t[0] = 0
    for k in range(1, m):
        t[k] = 1
    for k in range(m, n+1):
        t[k % m] = t[k % m] + t[(k-1) % m]
    return t[n % m]
```

Exercice 2

Question 5. La fonction ci-dessous repose sur le fait que si `n` représente un entier naturel non nul, alors `n % 10` retourne son dernier chiffre et `n // 10` ce nombre amputé de son dernier chiffre.

```
def somme_carre(n):
    s = 0
    while n > 0:
        s += (n % 10)**2
        n //= 10
    return s
```

Question 6. On définit alors :

```
def heureux(n):
    while True:
        if n == 1:
            return True
        elif n == 42:
            return False
        else:
            n = somme_carre(n)
```

Pourquoi choisir 42 et pas un autre nombre parmi la séquence 89, 145, 42, 20, 4, 16, 37, 58 ? Si vous vous posez la question c'est que vous n'êtes pas un vrai *geek* !

Exercice 3

Question 7. Si c est un carré alors $c = c_0c_1\cdots c_{p-1}c_p\cdots c_{n-1}$ avec $p = n/2$ et $c_0c_1\cdots c_{p-1} = c_pc_{p+1}\cdots c_{n-1}$. D'où la fonction :

```
def est_un_carre(c):
    p = len(c) // 2
    return c[:p] == c[p:]
```

Cette fonction présente le désavantage de masquer les comparaisons entre caractères individuels. On pourra donc lui préférer :

```
def est_un_carre(c):
    if len(c) % 2 != 0:
        return False
    p = len(c) // 2
    for i in range(p):
        if c[i] != c[p+i]:
            return False
    return True
```

Lorsque c est effectivement un carré, le nombre de comparaisons nécessaires pour s'en rendre compte est égal à $p = n/2$.

Question 8. Un facteur propre de $s = s_0s_1\cdots s_{n-1}$ est de la forme $s_is_{i+1}\cdots s_{j-1}$ avec $0 \leqslant i < j \leqslant n$. On les examine tous à la recherche d'un carré parmi eux :

```
def contient_un_carre(s):
    n = len(s)
    for i in range(n):
        for j in range(i+1, n+1):
            if est_un_carre(s[i:j]):
                return True
    return False
```

Compte tenu de la question précédente, le nombre de comparaisons entre caractères individuels peut être majoré par :

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n}\frac{j-i}{2} = \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=1}^{n-i} j = \frac{1}{4}\sum_{i=0}^{n-1}(n-i)(n-i+1) = \frac{1}{4}\sum_{i=1}^{n} i(i+1) = \frac{n(n+1)(n+2)}{12}.$$