TD n° 1 : Preuve d'algorithmes

EXERCICE 1 Justifier la terminaison et la correction des algorithmes ci-dessous. On énoncera précisément et on prouvera les variants et invariants utilisés.

1. Fonction qui détermine le rang du dernier terme strictement positif de la suite récurrente définie par $u_{n+1} = \frac{1}{2}u_n - 3n$ de premier terme $u_0 > 0$ donné en argument.

PYTHON
```
def rang_dernier_strictement_positif(u_0):
    u = u_0
    n = 0
    while u > 0:
        u = u/2 - 3*n
        n += 1
    return n - 1
```

2. Fonction qui calcule l'exponentiation de manière naïve.

PYTHON
```
def puissance(k, n):
    res = 1
    while n > 0:
        res = k * res
        n -= 1
    return res
```

3. Fonction qui calcule le logarithme itéré défini pour $x > 0$ par :

$$\log^*(x) = \min\left\{i \geq 0 \mid \log_2^{(i)}(x) \leq 1\right\}$$

PYTHON
```
def log_itere(x):
    k = 0
    while x > 1:
        x = math.log2(x)
        k += 1
    return k
```

4. Fonction qui vérifie l'appartenance d'un élément à une liste.

PYTHON
```
def appartient(element, liste):
    n = len(liste)
    i = 0
    while i < n and liste[i] != element:
        i += 1
    return i < n
```

5. Même chose mais avec une boucle inconditionnelle.

PYTHON
```
def appartient(element, liste):
    for autre in liste:
        if autre == element:
            return True
    return False
```

EXERCICE 2 Écrire en PYTHON avec une boucle `for` sur les indices, puis avec une boucle `while`, et justifier dans les deux cas la terminaison et la correction, des fonctions :

1. `maximum(liste)` qui renvoie la valeur maximale d'une liste non vide ;
2. `nb_occurrences(element, liste)` qui compte le nombre d'occurrences d'un élément dans une liste ;
3. `miroir(mot)` qui renvoie le mot miroir (*i.e.* lu à l'envers) d'une chaîne de caractères. Comment écrire cette fonction en une ligne avec des tranches ?
4. `fibonacci(n)` qui renvoie le terme d'indice n de la suite de Fibonacci ;
5. `est_fibonacci(n)` qui vérifie si l'entier n appartient à la suite de Fibonacci (boucle `while` seulement).