Licence Informatique — **UE INF152**
Licence Mathématiques — **Algorithmes et programmes**
Université Bordeaux 1 — **Devoir surveillé**
2009-2010

Corrigé du devoir du 11 mars 2010
Durée 1h30
Documents non autorisés

Exercice 1. Écrire une fonction `maximum(t)` qui retourne la plus grande valeur contenue dans un tableau `t` de `n` nombres entiers.

```
def maximum(t):
    if len(t) == 0:
        return None
    maxi = t[0]
    for i in range(1, len(t)):
        if t[i] > maxi:
            maxi = t[i]
    return maxi
```

Exercice 2. On se propose d'écrire une fonction `renverse(t)` pour renverser le contenu d'un tableau. Par exemple si on passe en paramètre à la fonction `renverse` le tableau `t = [1, 2, 3, 4]`, après l'appel `t` contiendra `[4, 3, 2, 1]`.
La fonction suivante n'atteint pas l'objectif fixé :

```
def renverse(t):
    taille = len(t)
    for i in range(taille):
        aux = t[i]
        t[i] = t[taille-i-1]
        t[taille-i-1] = aux
```

1. Vérifiez-le en observant le résultat de l'appel sur le tableau `t = [1, 2, 3, 4]`.
   Quel est le contenu de `t` après chaque tour de boucle ?

```
i = 0 -> t = [4, 2, 3, 1]
i = 1 -> t = [4, 3, 2, 1]
i = 2 -> t = [4, 2, 3, 1]
i = 3 -> t = [1, 2, 3, 4]
```

2. Proposez une modification de la fonction `renverse` pour renverser effectivement le tableau.

```
def renverser(t):
    taille = len(t)
    for i in range(taille // 2):
        aux = t[i]
        t[i] = t[taille-i-1]
        t[taille-i-1] = aux
```

3. Un tableau est dit “palindrome” si on lit la même suite de nombres en le parcourant de gauche à droite ou de droite à gauche. Par exemple, `[6, 2, 7, 4, 7, 2, 6]` est un tableau “palindrome”, `[2, 0, 0, 2]` également.

   Écrire une fonction `estPalindrome(t)` qui retourne `True` ou `False` suivant que `t` est un tableau “palindrome” ou pas.

```
def estPalindrome(t):
    taille = len(t)
    for i in range(taille // 2):
        if t[i] != t[taille-i-1]:
            return False
    return True
```

Exercice 3. Lors d'une course à vélo composée de `n` étapes, on s'intéresse aux altitudes des différentes lignes d'arrivée et de la ligne du départ de la course. Ces altitudes sont mémorisées dans un tableau `A` : `A[0]` représente l'altitude du point de départ et `A[1]` l'altitude du point d'arrivée de la première étape, `A[2]` représente l'altitude du point d'arrivée de la deuxième étape et ainsi de suite jusqu'à `A[n]`, altitude du point d'arrivée de la dernière étape.
Chaque étape est soit “montante” soit “descendante” et on appelle dénivelé l'écart d'altitude entre deux étapes (un dénivelé est toujours positif que l'étape soit montante ou descendante).

1. Écrire une fonction `nombreEtapesMontantes` qui prend en paramètre un tableau d'altitudes et calcule le nombre d'étapes “montantes”.

```
def nombreEtapesMontantes(t):
    nbMontantes = 0
    for i in range(1,len(t)):
        if t[i] >= t[i-1]:
            nbMontantes += 1
    return nbMontantes
```

   Remarque : Dans cette fonction, on a considéré montante ”au sens large”, c'est à dire qu'une étape à dénivelé nul est montante. On aurait aussi pu considérer montante au sens strict en remplaçant `"if t[i] >= t[i-1]:"` par `"if t[i] > t[i-1]:"`

2. Écrire une fonction `sommeDeniveles` qui calcule la somme de tous les dénivelés.

```
def sommeDeniveles(t):
    s = 0
    for i in range(1,len(t)):
        s += abs(t[i]-t[i-1])
    return s
```

Exemple :
pour le tableau `A = [500, 1100, 1200, 800, 200, 300]` le résultat de `nombreEtapesMontantes` est 3 (il y a trois étapes montantes : les deux premières et la dernière) et la fonction `sommeDeniveles` retourne 1800 (600 + 100 + 400 + 600 + 100).

Exercice 4. Soit la fonction :

```
def mystereRec(a,b):
    print(a,b)
    r = a%b
    if r == 0:
        return b
    return mystereRec(b,r)
```

1. Qu'affiche `mystereRec(5,2)` ?

```
5    2
2    1
```

Qu'affiche `mystereRec(45,10)` ?

```
45    10
10     5
```

2. Quelles sont les valeurs retournées par les appels de `mystereRec(5,2)` et `mystereRec(45,10)` ?

```
1
5
```

Exercice 5. La fonction `fibonacciIterative` ci-dessous calcule f_n, le n-ième terme ($n \geq 0$) de la suite de Fibonacci :

```
def fibonacciIterative(n):
    fib2 = 1
    fib1 = 1
    fib = 1
    k = 1
    while k < n :
        fib = fib2 + fib1
        fib2 = fib1
        fib1 = fib
        k += 1
    return fib
```

En utilisant `fibonacciIterative` on définit une nouvelle fonction `sommeFibo(x)` qui retourne la somme des `x+1` premiers nombres de la suite de Fibonacci :

```
def sommeFibo(x):
    somme = 0
    for i in range(x+1):
        somme += fibonacciIterative(i)
    return somme
```

1. Quels sont les résultats des appels `sommeFibo(0)`, `sommeFibo(1)`, `sommeFibo(4)` ?

```
sommeFibo(0) retourne 1
sommeFibo(1) retourne 2
sommeFibo(4) retourne 12
```

2. Compter le nombre d'additions effectuées lors de l'appel de `fibonacciIterative(n)`. Utiliser ce résultat pour déterminer le nombre d'additions effectuées lors de l'exécution de `sommeFibo(x)`.

```
Pour fibonacciIterative(n):
        0 addition si n=0
        2(n-1) additions si n > 0
Pour sommeFibo(x) :
        x * x + 1 additions
```

détail du calcul pour sommeFibo(x) :
pour x = 0, 1 addition
pour x > 0, $x + 1 + \sum_{i=1}^{x} 2(i-1) = x + 1 - 2x + 2\frac{x(x+1)}{2} = x^2 + 1$

3. Proposer une fonction `sommeFiboBis(x)` qui calcule le même résultat que la fonction `sommeFibo(x)` mais diminue le nombre d'opérations effectuées en évitant l'appel à `fibonacciIterative` pour chaque tour de boucle.

```
def sommeFiboBis(x):
    fib2 = 0
    fib1 = 1
    fib = 1
    somme = 0
    for i in range(x+1):
        somme += fib
        fib = fib2 + fib1
        fib2 = fib1
        fib1 = fib
    return somme
```

autre solution

```
def sommeFiboBis(x):
    fib2 = 1
    fib1 = 1
    fib = 1
    somme = 1
    for i in range(1,x+1):
        somme += fib
        fib = fib2 + fib1
        fib2 = fib1
        fib1 = fib
    return somme
```

Remarque : Le nombre d'additions effectuées lors de l'exécution de cette version de **sommeFiboBis(x)** est **2x**