Informatique PCSI
TP 9a corrigé : écriture d'un programme

Exercice 1

Avec l'appel `f([0])`, la variable locale `liste` prend la valeur `[0]`.

Avec l'affectation `liste = 2 * liste + m`, une nouvelle liste est créée. La liste `m` n'est pas copiée, et le comportement qui suit est identique à celui que l'on obtiendrait avec une affectation `liste = m`. Donc la modification du dernier élément de `liste` modifie le dernier élément de `m`.

Exercice 2

Un nombre `n` a une valeur entière si `n % 2` vaut 0 ou 1. C'est l'objet de la deuxième assertion.

```
def syracuse2(n):
    assert n > 0
    assert n % 2 == 0 or n % 2 == 1
    liste = [n]
    while n != 1:
        if n % 2:
            n = 3 * n + 1
        else:
            n = n // 2
        liste.append(n)
    return liste
```

On peut aussi utiliser l'assertion `assert n == int(n)` qui permet de vérifier si la valeur de `n` est entière ou pas.

On peut gérer le problème de valeur entière en modifiant l'instruction `if n % 2`. En effet ce test est passé pour tout flottant qui n'est pas pair. Il vaut mieux écrire à la place `if n % 2 == 1`.

Exercice 3

La variable `dico` peut être modifiée à chaque appel de la fonction `fibo`. Cependant grâce aux propriétés d'un dictionnaire, cette variable ne contient que les nombres de Fibonacci calculés par l'exécution de la fonction et ne contient chaque nombre calculé qu'une seule fois.

Exercice 4

Les variables `liste1` et `liste2` sont modifiées à chaque appel des fonctions. Dans les deux cas, la première utilisation de la fonction produit un résultat exact. Mais avec deux appels consécutifs comme `fibo1(3)` puis `fibo1(5)`, ou comme `fibo2(3)` puis `fibo2(5)`, le deuxième résultat est incorrect.

Pour obtenir un code sans effet de bord, la liste utilisée doit être définie dans le code de la fonction.

Voici un exemple de code :

```
def fibo(n):
    liste = [0, 1]
    for i in range(2, n+1):
        liste.append(liste[i-2] + liste[i-1])
    return liste[n]
```

Exercice 5

Il suffit d'obtenir les longueurs des deux listes avec la fonction `len` et de les comparer.

```
def ajoute(liste1, liste2):
    n1 = len(liste1)
    n2 = len(liste2)
    assert n1 == n2
    return [liste1[i] + liste2[i] for i in range(n1)]
```