Informatique PCSI
Corrigé TP 4

Exercice 1

1. Programme de base :

```
from random import randint

def devine(n):
    nombre = randint(1, n)
    choix = -1
    while choix != nombre:
        choix = int(input("Entrer un nombre: "))
        if choix < nombre:
            print("Trop petit")
        elif choix > nombre:
            print("Trop grand")
        else:
            print("Gagné !")
```

La stratégie optimale permettant de gagner avec le minimum de coups utilise la dichotomie.

2. Avec un compteur, on ajoute une variable `cpt` pour compter les coups. On l'initialise avec `cpt = 0` avant la boucle `while` et on l'incrémente après chaque choix du joueur par `cpt = cpt + 1`. On peut afficher sa valeur dans l'affichage final `print("Gagné en", cpt, "coups")`.

On peut aussi ajouter une instruction comme `print("Nombre de coups joués:", cpt)` ou, si `cmax` est le nombre maximal de coups, `print("Il vous reste", cmax-cpt, "coups à jouer")`. On ajoute alors la condition `cpt < cmax` pour la boucle while et l'affichage de "Perdu" si `cpt > cmax`.

Exercice 2

Plusieurs types de tests sont à envisager avec des fonctions simples comme la fonction identité ou la fonction carrée. Tester les cas où la solution est a, b, $(a+b)/2$.

Quelques exemples de tests qui montrent un problème :

```
def f(x):
    return x

def g(x):
    return x ** 2 - 9

print(dicho(f, -1, 2, 0.01))
print(dicho(f, -3, 3, 0.01))
print(dicho(f, 0, 3, 0.01))
print(dicho(g, 0, 10, 0.01))
print(dicho(g, 0, 6, 0.01))
```

Les deux premiers affichages sont -0.0009765625 et 2.9970703125. Dans le premier cas, on approche bien la solution 0 et dans le second cas on approche la borne 3. Il y a donc une erreur dans le programme.

À chaque passage dans la boucle, la solution x_0 doit appartenir à $[a; b]$. Or, si $f(a)$ vaut 0, lors d'un passage dans la boucle, a prend la valeur m et la solution n'appartient plus à $[a; b]$.

On peut corriger le code en remplaçant `f(a) * f(m) < 0` par `f(a) * f(m) <= 0`

Exercice 3

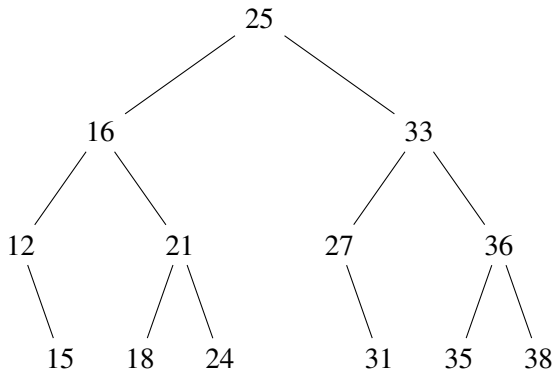
1. Pour la recherche de la valeur 6, les valeurs successives de `g` et `d` sont : 0 et 3 au début ; `k` prend la valeur 1 donc nous avons ensuite 1 et 3 ; `k` prend la valeur 2 donc nous avons 2 et 3 et la sortie de la boucle. Il en est de même pour les valeurs successives de `g` et `d` dans la recherche de la valeur 7. Seule la conclusion est différente.
2. Si la valeur initiale de `d` ne change pas, ceci signifie que la valeur cherchée est supérieure ou égale à la valeur de `liste[d-1]`. C'est la valeur de `g` qui est modifiée à chaque passage dans la boucle jusqu'à obtenir `g = d-1`. Et `liste[d]` n'existe pas.

   Si la valeur initiale de `d` est modifiée, cela intervient après une comparaison `x < liste[k]`. Et `d` prend alors la valeur de `k`. Donc on a bien `x < liste[d]` et la réponse est non.

Exercice 4

Au début, la valeur de l'indice gauche `g` est 0 et celle de l'indice droit `d` est 12. L'indice de `m` est donc 6 et correspond à l'élément 25.

Au niveau suivant on prend le « milieu » de la partie gauche, (indices 0 et 5), soit l'indice 2 qui correspond à l'élément 16 et le milieu de la partie droite, (indice 7 et 12), soit l'indice 9 qui correspond à l'élément 33. On continue ainsi de suite.



Exercice 5

1. Nous remplaçons la condition `g <= d` par la condition `g < d`.

   Considérons la liste `[2, 4]`. Nous cherchons l'élément `x` qui a pour valeur 4.

   Les valeurs initiales de `g` et `d` sont 0 et 1. La condition `g < d` est satisfaite, et donc `m` prend la valeur 0. C'est ensuite la condition `x > liste[m]` qui est satisfaite, donc `g` prend la valeur de `m+1`, soit 1. Les valeurs de `g` et `d` sont alors identiques et on sort de la boucle `while`. Donc la fonction renvoie `False` alors qu'elle doit renvoyer `True`.

   Avec la condition `g <= d`, on reste dans la boucle `while`. La variable `m` prend la valeur 1, puis la condition `x == liste[m]` est satisfaite et la fonction renvoie `True`.
2. Nous remplaçons l'affectation `g = m + 1` par l'affectation `g = m`.

   Considérons à nouveau la liste `[2, 4]` et nous cherchons l'élément `x` qui a pour valeur 4.

   Les valeurs initiales de `g` et `d` sont 0 et 1. La condition `g <= d` est satisfaite, donc `m` prend la valeur 0. C'est ensuite la condition `x > liste[m]` qui est satisfaite, donc `g` prend la valeur de `m` soit 0. Les valeurs de `g` et `d` n'ont pas changé et le programme entre dans une boucle infinie.

3. Nous remplaçons l'affectation `d = m - 1` par l'affectation `d = m`.

   Nous considérons encore la liste `[2, 4]` mais nous cherchons cette fois un élément `x` qui a pour valeur 3 et qui n'est donc pas présent dans la liste.

   Les valeurs initiales de `g` et `d` sont 0 et 1. La condition `g <= d` est satisfaite, donc `m` prend la valeur 0. C'est ensuite la condition `x > liste[m]` qui est satisfaite, donc `g` prend la valeur de `m+1`, soit 1. La condition `g <= d` est encore satisfaite, donc `m` prend la valeur 1. C'est ensuite la condition `x < liste[m]` qui est satisfaite, donc `d` garde la valeur 1 et le programme entre dans une boucle infinie.

4. La situation `g > d` peut se produire dans deux cas :
   - soit après l'affectation `g = m+1` si `g = m = d`;
   - soit après l'affectation `d = m-1` si `g = m = d` ou si `g = m = d-1`.

   Nous construisons un exemple pour chaque cas qui doit conduire à `g = m = d` ou `g = m = d-1`.
   - Premier cas : nous sommes dans le cas `g = m = d`.
     L'instruction `g = m+1` est exécutée si `x > liste[g]`, et donc `x > liste[d]`. Ceci signifie que `x` est strictement compris entre `liste[m]` et `liste[m+1]`, donc n'est pas dans la liste. Nous obtenons donc à la fin : `liste[d] < x < liste[g]`.
     Exemple avec la liste `[2, 4, 6]` et la recherche de la valeur 3 : les valeurs successives du couple d'indices (`g`, `d`) sont (0, 2) et `m` prend la valeur 1, puis (0, 0) et `m` prend la valeur 0 (nous en sommes à `g = d = m`), et enfin (1, 0). La fonction revoie `False`.
   - Deuxième cas : l'instruction `d = m-1` est exécutée si `x < liste[d]`, et donc `x < liste[g]`.
     Ceci signifie que `x` est strictement compris entre `liste[m-1]` et `liste[m]`, donc n'est pas dans la liste. Nous avons donc à la fin : `liste[d] < x < liste[g]`.
     Exemple avec la liste `[2, 4]` et la recherche de la valeur 3 : les valeurs successives du couple d'indices (`g`, `d`) sont (0, 1) et `m` prend la valeur 0, puis (1, 1) et `m` prend la valeur 1 (nous en sommes à `g = d = m`), et enfin (1, 0). La fonction revoie `False`.
     Avec la liste `[2, 4, 6, 8]` et la recherche de 5 : les valeurs successives du couple d'indices (`g`, `d`) sont (0, 3) et `m` prend la valeur 1, puis (2, 3) et `m` prend la valeur 2 (nous en sommes à `g = d-1 = m`), et enfin (2, 1). La fonction revoie `False`.

Exercice 6

```
def insertion(liste, x):
    liste.append(x)
    n = len(liste)
    g, d = 0, n-1
    while g <= d:
        m = (g + d) // 2
        if x < liste[m]:
            d = m - 1
        else:
            g = m + 1
    for j in range(g+1, n):
        liste[n+g-j] = liste[n+g-j-1]
    if g < n:
        liste[g] = x
```