Correction du DS 1

Informatique pour tous, première année

Julien Reichert

Exercice 1

Le script (a) provoque une erreur lors de l'appel de la fonction car la variable `i` n'est pas définie (elle pourrait être globale, mais le script se limitant aux lignes écrites ici, l'erreur est inévitable).

Le script (b) provoque une erreur à deux titres lors de l'appel de la fonction : en tant qu'entier, `42` ne supporte pas la fonction `len`[1], et quand bien même l'entrée serait par exemple une liste, l'élément `a[len(a)]` auquel on tente d'accéder au premier tour dans la boucle n'existe pas et cause un dépassement.

Le script (c) provoque une erreur lors de l'exécution du script. En effet, la ligne `return 1`, n'étant pas indentée, n'est pas dans la définition de la fonction, et un `return` en-dehors d'une fonction cause une erreur.

Le script (d) est correct. On notera qu'à la fin d'une boucle inconditionnelle, la variable de boucle existe tout de même et vaut la dernière valeur parcourue, ce qui peut être utile si la boucle est interrompue par un `break`[2].

Le script (e) provoque une erreur lors de l'exécution du script. Là aussi, la ligne `print(n)` est hors de la définition de la fonction, et `n` n'est pas définie (ce qui aurait sauvé les meubles).

Exercice 2

```
def deuxieme_plus_grand(liste):
    if len(liste) < 2:
        raise ValueError("Liste trop courte") # de toute façon il y aurait une erreur automatique
    if liste[0] < liste[1]:
        premier = liste[1]
        deuxieme = liste[0]
    else:
        premier = liste[0]
        deuxieme = liste[1]
    for i in range(2,len(liste)):
# on ne doit pas retester les premiers, au cas où il y aurait le maximum
        if liste[i] >= premier: # pas > pour économiser un test dans certains cas
            deuxieme = premier
            premier = liste[i] # dans cet ordre absolument
        elif liste[i] > deuxieme:
# donc liste[i] < premier, on ne met pas >= pour économiser les affectations en cas d'égalité
            deuxieme = liste[i]
    return deuxieme
```

1. On dit qu'il n'est pas *sized*.
2. à éviter dans les programmes, si possible

Un algorithme optimal se contente d'un parcours de la liste en entrée avec au plus deux comparaisons, plutôt que deux parcours (dans lesquels il y aurait une double comparaison, ce qui masque le fait que la complexité soit quoi qu'il en soit doublée). Attention aux erreurs classiques : l'initialisation des variables représentant le maximum ne peuvent se faire à la valeur `0`, par exemple, car rien n'exclut que la liste comporte des valeurs positives. Au passage, le programme ci-dessus tolère que l'entrée soit un *indexable sized* quelconque dont les éléments sont comparables entre eux sans nécessairement être des nombres. En outre, la spécification précise demande que si le maximum apparaît deux fois, c'est cette valeur qui sera retournée [3].

Exercice 3

La fonction `mystere1` a deux arguments : `a` pour lequel les `a[i]` indiquent qu'il est *indexable*, par exemple une liste ou une chaîne de caractères, et `n` qu'on passe aussi en argument de la fonction `int`, donc `n` est un entier, un flottant ou une chaîne de caractères (composée uniquement de chiffres si on veut éviter une erreur). La boucle qu'on définit parcourt exactement les `int(n)` premiers éléments de `a`, ce qui veut dire que rien ne se passe si `int(n)` $\leq$ `0`, mais surtout **qu'on a une erreur s'il est supérieur à la taille de** `a`. Au vu du test effectué dans le corps de la boucle, la valeur de retour est le plus grand des éléments parcourus, de type `NoneType` (la valeur étant `None`) si rien n'est parcouru ou dans le cas pathologique où `a` commencerait par `int(n)` occurrences de `None` [4], et sinon d'un type un peu quelconque, car `a` ne contient pas forcément que des entiers ou même des nombres en général, le tout est que les éléments doivent supporter la comparaison entre eux (à l'exception des `None` initiaux, donc).

La fonction `mystere2` a un argument : `n`, qui comme son nom ne l'indique pas n'est pas un entier mais un *sized*, là aussi les listes et les chaînes de caractères en sont des bons exemples. La variable `s`, définie dans la fonction, au vu de son initialisation et de ce qu'on y ajoute [5], est une chaîne de caractères qui contient à la fin `"0123...k"` où `k` est la taille de `n` moins un en tant que chaîne de caractères. Attention, la valeur de retour de la fonction est `None` puisque `s` n'est pas retournée mais **imprimée**.

3. et non pas imprimée !
4. Ben oui, pourquoi pas ?
5. On dit « concatène » pour des chaînes de caractères.