Correction du DS 3

Julien Reichert

Exercice 1

Une fois la fonction définie, le programme évalue la condition de la distinction de cas, et lance la fonction `test` sur l'argument `"a"`, ce qui l'imprime sur une ligne et retourne `False` car la taille n'est pas strictement supérieure à un.

La condition n'est pour le moment pas réalisée, d'où l'évaluation de la suite, en l'occurrence la fonction `test` sur l'argument `"ab"`, lui aussi imprimé sur une nouvelle ligne. Cette fois-ci, la fonction `test` retourne `True`, et la condition devient vérifiée car il s'agit d'une expression booléenne formée par deux disjonctions (`or`) consécutives.

Dans la mesure où l'évaluation des opérateurs booléens de base est paresseuse, le troisième appel à `test` est ignoré, donc la chaîne `"b"` ne sera pas imprimée et on entre directement dans le corps du `if`, qui imprime `"c"` sur une autre ligne.

Exercice 2

Le script (a) ne provoque pas d'erreur et imprime les entiers de `0` à `41` sur des lignes successives de la console.

Le script (b) ne provoque pas d'erreur mais il n'imprime rien car l'objet `range` est vide (le premier paramètre de la fonction qui l'engendre est supérieur au deuxième sans qu'un troisième paramètre strictement négatif ne soit fourni).

Le script (c) provoque une erreur lors de l'exécution du script. En effet, la ligne `return n`, 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) provoque une erreur lors de l'appel de la fonction. En effet, le paramètre `l` est une liste, et il est interdit d'appeler la fonction `range` sur autre chose qu'un entier.

Le script (e) ne provoque pas d'erreur à proprement parler, mais il déclenche une boucle infinie imprimant un nombre illimité de lignes contenant `42`. En effet, la ligne `n-1` ne provoque pas de modification de la variable `n`, ce qui fait que le test d'entrée dans la boucle est toujours vérifié.

Le script (f) ne provoque pas d'erreur, malgré le test qui devrait faire faire une division par zéro. En fait, le premier tour de la boucle déclenche la fin de la fonction en raison du `return`, et seul `0` sera donc retourné, et éventuellement imprimé dans la console si c'est là que l'appel avait été effectué.

Exercice 3

```
def uniques(s, t):
    rep = ""
    for c in s:
        if c not in t:
            rep += c
    for c in t:
        if c not in s:
            rep += c
    return rep
```

L'opérateur `in` est un test d'appartenance, qui remplace une boucle sur la séquence dans laquelle l'appartenance est testée. Il en va de même quand on ajoute un `not`. Ainsi, le nombre de comparaisons dans le pire des cas est le nombre d'éléments dans une chaîne pour chaque élément dans l'autre, ceci pour chacune des boucles, pour un total en $\mathcal{O}$(`len(s).len(t)`). Le nombre d'ajouts, quant à lui, est d'un ou zéro pour chaque élément des deux chaînes, il sera donc linéaire.

Exercice 4

```
def somme_consecutifs(l):
    rep = []
    actuel = l[0]
    occurrences = 0
    for x in l + [None]: # Ajouter un tour est une astuce pour économiser un copier-coller
        if x == actuel:
            occurrences += 1
        else:
            rep.append(actuel*occurrences)
            actuel = x
            occurrences = 1
    return rep
```

Exercice 5

```
def somme_successifs(l):
    if l == []:
        return []
    rep = [l[0]]
    for i in range(1, min(7, len(l))):
        rep.append(rep[-1] + l[i])
    for i in range(7, len(l)):
        rep.append(rep[-1] + l[i] - l[i-7])
    return rep
```

Exercice 6

```
def sfisseccus_emmos(l):
    if l == []:
        return []
    rep = [l[0]]
    for i in range(1, min(7, len(l))):
        rep.append(l[i] - l[i-1])
    for i in range(7, len(l)):
        rep.append(l[i] - l[i-1] + rep[i-7])
    return rep
```

Exercice 7

```
def motif_commun(l, ll, k):
    for i in range(len(l)-k+1):
        for j in range(len(ll)-k+1):
            if l[i:i+k] == ll[j:j+kk]:
                return l[i:i+k]
```

Exercice 8

```
def tous_lancers(n):
    lancer = [0] * n
    lancers = []
    while lancer[0] < 6:
        lancers.append(lancer[:]) # Ne pas oublier la copie !
        j = n-1
        lancer[j] += 1
        while j > 0 and lancer[j] == 6:
            lancer[j] = 0
            j -= 1
            lancer[j] += 1
    return lancers
```

Une autre méthode intéressante consiste à renvoyer la liste des chiffres en base `6` de tous les nombres de `0` à `6**n-1`...