NSI_21_sujet_8

September 3, 2021

1 Correction et prolongements du sujet 8

Les prolongements n'étaient évidemment pas exigés à l'examen

1.1 Exercice 1

[25]:
```
def recherche(caractere: str, mot: str) -> int:
    """
    renvoie le nombre d'occurences de 'caractere' dans la chaine éventuellement␣
 ↪vide 'mot'.
    caractere, mot: chaînes (str)
    """
    nb_oc = 0
    for c in mot:
        if c == caractere:
            nb_oc += 1
    return nb_oc
```

[26]:
```
recherche('e', 'sciences')
```

[26]: 2

[27]:
```
recherche('i', 'mississippi')
```

[27]: 4

[28]:
```
recherche('a', 'mississippi')
```

[28]: 0

1.2 Versions récursives (prolongement)

1.2.1 Version utilisant le “slicing”

[33]:
```
def recherche_f1(c, m, acc=0):
    """
    renvoie le nombre d'occurences de c dans la chaine éventuellement vide m.
    c, m: chaînes (str)
    acc: accumulateur (int), contient le nombre d'occurences
    """
    if m == '':
        return acc
    else:
        if c == m[0]:
            return recherche_f1(c, m[1:], acc + 1)
        else:
            return recherche_f1(c, m[1:], acc)
```

[34]:
```
recherche_f1('e', 'sciences')
```

[34]: 2

[35]:
```
recherche_f1('i', 'mississippi')
```

[35]: 4

[36]:
```
recherche_f1('a', 'mississippi')
```

[36]: 0

1.2.2 Version sans slicing

[39]:
```
def recherche_f2(c, m, i=0, acc=0):
    """
    renvoie le nombre d'occurences de c dans la chaine éventuellement vide m.
    c, m: chaînes (str).
    acc: accumulateur (int), contient le nombre d'occurences;
    i: index (int) d'un caractère dans m.
    """
    if i >= len(m):
        return acc
    else:
        if c == m[i]:
            return recherche_f2(c, m, i+1, acc + 1)
        else:
            return recherche_f2(c, m, i+1, acc)
```

[40]:
```
recherche_f2('e', 'sciences')
```

[40]: 2

[41]:
```
recherche_f2('i', 'mississippi')
```

[41]: 4

[42]:
```
recherche_f2('a', 'mississippi')
```

[42]: 0

1.3 Qu'en est-il des temps d'exécution ?

[60]:
```
import string
import random

# construisons une chaine "aléatoire" de 1000 caractères
ch = ''.join([random.choice(string.ascii_letters) for _ in range(1000)])
```

[57]:
```
%timeit recherche('a', ch)
```

29.8 µs ± 707 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

[58]:
```
%timeit recherche_f1('a', ch)
```

277 µs ± 8.15 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

[59]:
```
%timeit recherche_f2('a', ch)
```

221 µs ± 4.67 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

La version itérative est plus rapide d'un facteur 10 environ.

1.4 Exercice 2 (modifié)

[62]:
```
Pieces = [100, 50, 20, 10, 5, 2, 1]

def rendu_glouton(arendre, solution, i=0):
    if arendre == 0:
        return solution
    p = Pieces[i]
    if p <= arendre:
        solution.append(p)
        return rendu_glouton(arendre - p, solution, i)
    else:
```

```
        return rendu_glouton(arendre, solution, i+1)
```

[63]:
```
rendu_glouton(68, [], 0)
```

[63]:
```
[50, 10, 5, 2, 1]
```

[64]:
```
rendu_glouton(291, [], 0)
```

[64]:
```
[100, 100, 50, 20, 20, 1]
```

1.5 L'exercice en version originale (prolongement)

La proposition erronée du sujet renfermait une erreur subtile (en dehors du problème d'indentation ou du passage de paramètre lors du 2ème `if`).

[91]:
```
Pieces = [100, 50, 20, 10, 5, 2, 1]

def rendu_glouton_i(arendre, solution=[], i=0):
    if arendre == 0:
        return solution
    p = Pieces[i]
    if p <= arendre:
        solution.append(p)
        return rendu_glouton(arendre - p, solution, i)
    else:
        return rendu_glouton(arendre, solution, i+1)
```

[92]:
```
rendu_glouton_i(68, [], 0)
```

[92]:
```
[50, 10, 5, 2, 1]
```

[93]:
```
rendu_glouton_i(291, [], 0)
```

[93]:
```
[100, 100, 50, 20, 20, 1]
```

Le résultat semble correct. Mais essayons d'exploiter les paramètres par défaut…

[94]:
```
rendu_glouton_i(291)
```

[94]:
```
[100, 100, 50, 20, 20, 1]
```

[95]:
```
rendu_glouton_i(291)
```

[95]:
```
[100, 100, 50, 20, 20, 1, 100, 100, 50, 20, 20, 1]
```

… ce qui est manifestement faux ! Le problème vient du paramètre par défaut `solution=[]`. La documentation python précise:

Les valeurs par défaut des paramètres sont évaluées de la gauche vers la droite quand la définition de la fonction est exécutée. Cela signifie que l'expression est évaluée une fois, lorsque la fonction est définie, et que c'est la même valeur « pré-calculée » qui est utilisée à chaque appel. C'est particulièrement important à comprendre lorsqu'un paramètre par défaut est un objet mutable, tel qu'une liste ou un dictionnaire : si la fonction modifie l'objet (par exemple en ajoutant un élément à une liste), la valeur par défaut est modifiée.

En d'autres termes c'est le même objet *liste* qui est appelé à chaque exécution de la fonction.

D'une manière générale, il faut éviter de mettre des structures mutables comme valeur par défaut

Une solution issue de l'exploitation de la documentation

[96]:
```
Pieces = [100, 50, 20, 10, 5, 2, 1]

def rendu_glouton_ii(arendre, solution=None, i=0):
    if solution is None:
        solution = []
    if arendre == 0:
        return solution
    p = Pieces[i]
    if p <= arendre:
        solution.append(p)
        return rendu_glouton(arendre - p, solution, i)
    else:
        return rendu_glouton(arendre, solution, i+1)
```

[98]:
```
rendu_glouton_ii(291)
```

[98]:
```
[100, 100, 50, 20, 20, 1]
```

[99]:
```
rendu_glouton_ii(291)
```

[99]:
```
[100, 100, 50, 20, 20, 1]
```

[100]:
```
rendu_glouton_ii(68)
```

[100]:
```
[50, 10, 5, 2, 1]
```

[ ]:
```

```