Sujet Zéro – 2023 – sujet2 - Correction

Exercice 2 (4 points)

👍 En fin de correction vous trouverez une explication détaillée de l´exécution d'un algorithme récursif avec schéma de pile d'appels.

1. Questions préliminaires

a. Récursivité — Définition

Une fonction récursive est une fonction qui **s'appelle elle-même** au cours de son exécution.

👉 *Commentaire : Une fonction récursive comporte toujours un cas d'arrêt pour éviter une récursion infinie.*

---

b. Pourquoi le programme s'arrête après l'affichage de 0 ?

Le programme s'arrête car lorsque le paramètre vaut 0, la condition d'arrêt est atteinte et la fonction ne s'appelle plus elle-même.

👉 *Commentaire : Le cas de base (n == 0) empêche les appels récursifs supplémentaires et garantit la terminaison.*

---

2. Écriture d'une fonction récursive — Factorielle

La définition mathématique :

- $fact(0) = 1$
- $fact(n) = n \times fact(n-1)$

Programme complété :

```python
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

👉 *Commentaire : Le cas d'arrêt correspond à n == 0 ; chaque appel réduit la valeur de n.*

---

4. Analyse d'exécution — Somme des entiers récursive

On exécute :

```
res = somme_entiers_rec(3)
```

a. Affichage dans la console

L'instruction `print(n)` est exécutée à chaque appel récursif.

Les valeurs affichées seront :

```
3
2
1
0
```

👉 *Commentaire : Les affichages se font lors de la descente récursive.*

---

b. Valeur de la variable res

Calcul effectué :

0+1+2+3=60 + 1 + 2 + 3 = 60+1+2+3=6

Donc :

```
res = 6
```

👉 *Commentaire : Les additions se font lors de la remontée de la pile d'appels.*

---

4. Écriture d'une fonction non récursive — Somme des entiers

Version itérative :

```
def somme_entiers(n):
    total = 0
    for i in range(n + 1):
        total = total + i
    return total
```

👉 *Commentaire : La version itérative utilise une boucle plutôt qu'un empilement d'appels récursifs.*

BONUS : Comment fonctionne la pile d'appels récursifs ?

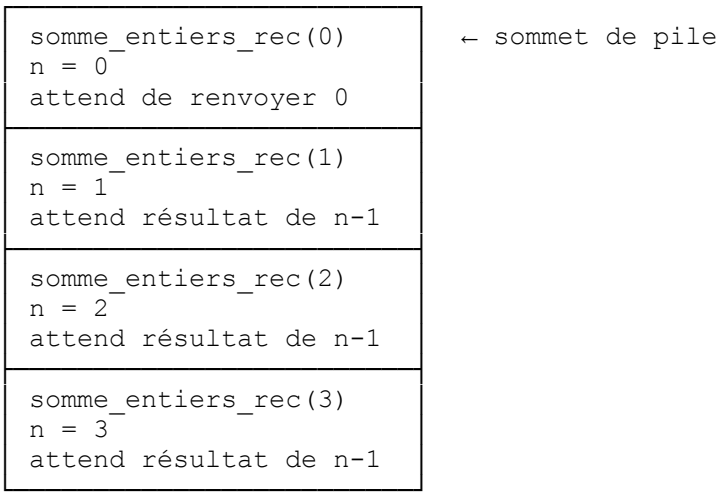
```
def somme_entiers_rec(n):
    if n == 0:
        return 0
    else:
        print(n) # pour vérification
        return n + somme_entiers_rec(n-1)
```

Étudions l'appel : `somme_entiers_rec(3)`

1. Phase descendante — Empilement des appels

À chaque appel récursif, un nouveau frame (cadre d'exécution) est ajouté à la pile.

Représentation visuelle au moment le plus profond (n=0):



👉 *Commentaire : Chaque appel reste en mémoire tant que l'appel suivant n'a pas renvoyé sa valeur.*

Affichage console pendant la descente :

```
3
2
1
0
```

---

2. Cas d'arrêt

La frame tout en haut correspond à : `somme_entiers_rec(0)`

Elle renvoie immédiatement : `0`

👉 *Commentaire : Le cas d'arrêt déclenche la phase de remontée.*

---

3. Phase montante — Dépilement et calcul

La pile commence à se vider lorsque le cas de base a été atteint.

🔷 Retour vers n = 1

```
1 + 0 = 1
```

Pile restante :

somme_entiers_rec(1)	→ renvoie 1
somme_entiers_rec(2)	
somme_entiers_rec(3)	

🔷 **Retour vers n = 2**

```
2 + 1 = 3
```

🔷 **Retour vers n = 3**

```
3 + 3 = 6
```

4. Pile complètement vidée

```
Pile vide
```

Résultat final : `res = 6`

👉 *Commentaire : Les additions se font uniquement lors du dépilement.*

Visualisation synthétique (descente / montée)

Descente (empilement)

```
3 → 2 → 1 → 0
```

Montée (calculs)

```
0 → 1 → 3 → 6
```
