Effets de bords

Corrigés

Exercice 1 : Gestion d'un compteur global

Dans cet exercice, vous allez implémenter une fonction qui utilise une variable globale comme compteur. Le but est de voir comment l'utilisation de variables globales peut introduire des effets de bord dans votre programme.

Instructions :

1. Déclarez une variable globale appelée compteur avec une valeur initiale de 0.
2. Créez une fonction appelée incrementer_compteur() qui :
    - Utilise la variable globale compteur.
    - Incrémente la valeur de compteur de 1 à chaque appel.
    - Affiche la valeur actuelle de compteur.
3. Créez une autre fonction appelée reset_compteur() qui remet la valeur de compteur à 0.
4. Appelez ces fonctions plusieurs fois pour observer comment la valeur de la variable globale est modifiée, même en dehors des fonctions.

Note : Utilisez Python Tutor afin de visualiser le mécanisme de variable globale: https://pythontutor.com

Exemple de résultat attendu :

```
# Initialisation
>>> compteur = 0

# Appels successifs de la fonction incrementer_compteur
>>> incrementer_compteur()
Compteur actuel : 1
>>> incrementer_compteur()
Compteur actuel : 2
>>> incrementer_compteur()
Compteur actuel : 3

# Remise à zéro du compteur
>>> reset_compteur()
Compteur remis à zéro : 0

# Appel de la fonction après remise à zéro
>>> incrementer_compteur()
Compteur actuel : 1
```

Programme principal à compléter :

```
# Déclaration d'une variable globale
compteur =

# Fonction pour incrémenter le compteur
def incrementer_compteur():


# Fonction pour réinitialiser le compteur
def reset_compteur():


# Test de l'exercice
incrementer_compteur()  # Affiche "Compteur actuel : 1"
incrementer_compteur()  # Affiche "Compteur actuel : 2"
reset_compteur()      # Affiche "Compteur remis à zéro : 0"
incrementer_compteur()  # Affiche "Compteur actuel : 1"
```

Exercice 2 : Aliassage et modification de listes

Dans cet exercice, tu vas manipuler des listes pour observer les effets de bord liés à l'aliasage. L'aliassage se produit lorsque plusieurs variables pointent vers le même objet en mémoire. En modifiant l'une de ces variables, l'objet sous-jacent est modifié, ce qui affecte toutes les autres variables qui font référence à cet objet.

Instructions :

1. Crée une liste liste_originale avec quelques éléments (par exemple [1, 2, 3, 4]).
2. Crée une nouvelle variable liste_alias qui **pointe vers la même liste** que liste_originale.
3. Modifie des éléments de liste_alias et observe les changements sur liste_originale pour comprendre l'aliasage.
4. Crée une copie **indépendante** de liste_originale dans une nouvelle variable liste_copie.
5. Modifie des éléments de liste_copie et montre que ces modifications **n'affectent pas** liste_originale.

Objectif pédagogique :

- **Effet de bord lié à l'aliassage** : L'aliassage se produit quand deux variables pointent vers le même objet en mémoire. Toute modification faite via l'une affecte également l'autre.
- **Copie superficielle** : En copiant une liste avec [:], tu crées une nouvelle liste indépendante. Les modifications faites à l'une n'affectent pas l'autre.

Note : Utilisez Python Tutor afin de visualiser le mécanisme d'aliassage: https://pythontutor.com

```
# Étape 1 : Création de la liste originale
liste_originale = [1, 2, 3, 4]

# Étape 2 : Création d'un alias (liste_alias pointe vers la même liste que liste_originale)
liste_alias = ...

# Modification de liste_alias
liste_alias[0] = ...

# Affichage des deux listes pour observer les effets de bord
print("Liste originale après modification via alias :", liste_originale)  # [99, 2, 3, 4]
print("Liste alias :", liste_alias)  # [99, 2, 3, 4]

# Étape 3 : Création d'une copie indépendante
liste_copie = ...

# Modification de la copie
liste_copie[1] = 88

# Affichage des listes pour observer l'absence d'effet de bord avec la copie
print("Liste originale après modification de la copie :", liste_originale)  # [99, 2, 3, 4]
print("Liste copie :", liste_copie)  # [99, 88, 3, 4]
```

Exercice 3 : Copie superficielle vs. copie profonde

Objectif :

Apprendre la différence entre une copie superficielle et une copie profonde, et comprendre comment ces concepts affectent les objets mutables imbriqués comme les listes de listes.

Instructions :

1. **Copie superficielle** : Créez une liste contenant d'autres listes. Réalisez une copie superficielle de cette liste et modifiez un des éléments dans l'une des sous-listes de la copie superficielle. Observez comment cela affecte la liste originale.
2. **Copie profonde** : Réalisez une copie profonde de la même liste (utilisez le module copy pour cela) et modifiez un élément dans l'une des sous-listes de la copie profonde. Observez comment cela affecte la liste originale.

Exemple de résultat attendu :

1. **Copie superficielle** :
    - Modifiez un élément dans une sous-liste de la copie superficielle.
    - La liste originale doit également refléter cette modification.
2. **Copie profonde** :
    - Modifiez un élément dans une sous-liste de la copie profonde.
    - La liste originale ne doit pas être affectée par cette modification.

Note : Utilisez Python Tutor afin de visualiser les mécanismes de copies : https://pythontutor.com

```
import ...

# Création de la liste originale contenant des sous-listes
liste_originale = [[1, 2, 3], [4, 5, 6]]

# Étape 1 : Copie superficielle
liste_copie_superficielle = ...

# Modification dans la copie superficielle
liste_copie_superficielle[0][0] = ...

# Affichage pour observer les effets de bord
print("Liste originale après modification de la copie superficielle :", liste_originale)
# Liste originale après modification de la copie superficielle : [[99, 2, 3], [4, 5, 6]]
print("Liste copie superficielle :", liste_copie_superficielle)
# Liste copie superficielle : [[99, 2, 3], [4, 5, 6]]

# Étape 2 : Copie profonde
liste_copie_profonde = ....

# Modification dans la copie profonde
liste_copie_profonde[1][0] = ...

# Affichage pour observer les effets de bord
print("Liste originale après modification de la copie profonde :", liste_originale)
# Liste originale après modification de la copie profonde : [[99, 2, 3], [4, 5, 6]]
print("Liste copie profonde :", liste_copie_profonde)
# Liste copie profonde : [[99, 2, 3], [88, 5, 6]]
```

Corrections

Exercice : Gestion d'un compteur global

```
# Déclaration d'une variable globale
compteur = 0

# Fonction pour incrémenter le compteur
def incrementer_compteur():
    global compteur  # On précise que l'on va utiliser la variable globale
    compteur += 1
    print(f"Compteur actuel : {compteur}")

# Fonction pour réinitialiser le compteur
def reset_compteur():
    global compteur
    compteur = 0
    print(f"Compteur remis à zéro : {compteur}")

# Test de l'exercice
incrementer_compteur()  # Affiche "Compteur actuel : 1"
incrementer_compteur()  # Affiche "Compteur actuel : 2"
reset_compteur()      # Affiche "Compteur remis à zéro : 0"
incrementer_compteur()  # Affiche "Compteur actuel : 1"
```

Exercice : Aliasage et modification de listes

```
# Étape 1 : Création de la liste originale
liste_originale = [1, 2, 3, 4]

# Étape 2 : Création d'un alias (liste_alias pointe vers la même liste que liste_originale)
liste_alias = liste_originale

# Modification de liste_alias
liste_alias[0] = 99

# Affichage des deux listes pour observer les effets de bord
print("Liste originale après modification via alias :", liste_originale)  # [99, 2, 3, 4]
print("Liste alias :", liste_alias)  # [99, 2, 3, 4]

# Étape 3 : Création d'une copie indépendante
liste_copie = liste_originale[:]

# Modification de la copie
liste_copie[1] = 88

# Affichage des listes pour observer l'absence d'effet de bord avec la copie
print("Liste originale après modification de la copie :", liste_originale)  # [99, 2, 3, 4]
print("Liste copie :", liste_copie)  # [99, 88, 3, 4]
```

Objectif pédagogique :

- **Effet de bord lié à l'aliasage** : L'aliasage se produit quand deux variables pointent vers le même objet en mémoire. Toute modification faite via l'une affecte également l'autre.
- **Copie superficielle** : En copiant une liste avec [:], vous créez une nouvelle liste indépendante. Les modifications faites à l'une n'affectent pas l'autre.

Exercice : Copie superficielle vs. copie profonde

```
import copy

# Création de la liste originale contenant des sous-listes
liste_originale = [[1, 2, 3], [4, 5, 6]]

# Étape 1 : Copie superficielle
liste_copie_superficielle = liste_originale.copy()

# Modification dans la copie superficielle
liste_copie_superficielle[0][0] = 99

# Affichage pour observer les effets de bord
print("Liste originale après modification de la copie superficielle :", liste_originale)
# Liste originale après modification de la copie superficielle : [[99, 2, 3], [4, 5, 6]]
print("Liste copie superficielle :", liste_copie_superficielle)
# Liste copie superficielle : [[99, 2, 3], [4, 5, 6]]

# Étape 2 : Copie profonde
liste_copie_profonde = copy.deepcopy(liste_originale)

# Modification dans la copie profonde
liste_copie_profonde[1][0] = 88

# Affichage pour observer les effets de bord
print("Liste originale après modification de la copie profonde :", liste_originale)
# Liste originale après modification de la copie profonde : [[99, 2, 3], [4, 5, 6]]
print("Liste copie profonde :", liste_copie_profonde)
# Liste copie profonde : [[99, 2, 3], [88, 5, 6]]
```

Explications :

1. **Copie superficielle** :
    - Crée une nouvelle liste, mais les sous-listes internes sont partagées avec l'originale.
    - Toute modification dans les sous-listes de la copie superficielle affecte également les sous-listes dans la liste originale.
2. **Copie profonde** (copy.deepcopy(liste_originale)):
    - Crée une nouvelle liste et également des nouvelles sous-listes (copie récursive de tous les niveaux).

- Les modifications dans les sous-listes de la copie profonde n'affectent pas les sous-listes dans la liste originale.