Python - Les opérateurs logiques - Évaluation paresseuse

L'évaluation paresseuse, appelée aussi short-circuit evaluation, est une technique utilisée par les langages de programmation comme Python pour optimiser l'exécution des expressions logiques. Elle consiste à arrêter l'évaluation d'une expression logique dès qu'il n'est plus nécessaire de continuer, en fonction du résultat déjà déterminé.

5GMS 6GMS Découverte

Évaluation Paresseuse (Short-Circuit Evaluation)

L'évaluation paresseuse, appelée aussi **short-circuit evaluation**, est une technique utilisée par les langages de programmation comme Python pour optimiser l'exécution des expressions logiques. Elle consiste à **arrêter l'évaluation d'une expression logique dès qu'il n'est plus nécessaire de continuer**, en fonction du résultat déjà déterminé.

Principe de base

L'évaluation paresseuse repose sur les règles fondamentales des opérateurs logiques `and` et `or` :

1. **Pour `and` :**
   - Si une condition est `False`, le résultat final sera forcément `False`, peu importe les autres conditions restantes. Il est inutile de continuer l'évaluation.
   - Exemple :

```
condition1 and condition2 and condition3
```

     - Si `condition1` est `False`, Python s'arrête et ignore `condition2` et `condition3`.
2. **Pour `or` :**
   - Si une condition est `True`, le résultat final sera forcément `True`, peu importe les autres conditions restantes. Il est inutile de continuer l'évaluation.
   - Exemple :

```
condition1 or condition2 or condition3
```

     - Si `condition1` est `True`, Python s'arrête et ignore `condition2` et `condition3`.

Avantages de l’évaluation paresseuse

1. **Amélioration des performances :**
   - L'évaluation paresseuse évite de calculer inutilement les conditions restantes, ce qui économise du temps et des ressources.
   - Cela est particulièrement utile si certaines conditions sont complexes ou coûteuses à évaluer (exemple : fonctions ou requêtes vers une base de données).
2. **Sécurité et gestion des erreurs :**
   - Permet d'éviter des erreurs en ne procédant pas à des évaluations inutiles.
   - Exemple : Protéger une division par zéro.

```python
a = 0
b = 5
if a != 0 and b / a > 1:  # Python n'exécute pas b / a si a == 0
    print("Condition remplie")
else:
    print("Condition non remplie")
```

Exemples Concrets

> Teste ces exemples avec Thonny en mode debug, en exécutant ces codes pas à pas (`F7`). Tu verras effectivement que toutes les conditions ne sont pas testées.

1. Évaluation avec `and`

```python
x = 5
y = 10

if x > 0 and y / x > 1:
    print("Conditions valides")
else:
    print("Au moins une condition est invalide")
```

Déroulement :

1. Python évalue `x > 0` → Résultat : `True`.
2. Passe à `y / x > 1` → Résultat : `True`.
3. Le résultat final est donc `True` et l'instruction `print("Conditions valides")` est exécutée.

Mais si `x = 0` :

1. Python évalue `x > 0` → Résultat : `False`.
2. Python s'arrête immédiatement sans évaluer `y / x` (prévenant une division par zéro).

---

2. Évaluation avec `or`

```python
def test():
    print("Fonction exécutée")
    return True

if True or test():
    print("Condition remplie")
```

Déroulement :

1. La première condition (`True`) est évaluée.
2. Python s'arrête immédiatement et **n'appelle pas** la fonction `test()`.

Résultat :

```
Condition remplie
```

La phrase `"Fonction exécutée"` n'apparaît pas.

---

3. Combinaison d'expressions

Un cas combinant des calculs et des protections contre des erreurs possibles.

```python
a = 0
b = 10

if a != 0 and (b / a > 5 or b > 20):
    print("Condition remplie")
else:
    print("Condition non remplie")
```

Déroulement :

1. Python commence par évaluer `a != 0`. Si c'est `False`, il s'arrête immédiatement, sans risquer une division par zéro.

Limites et précautions

1. **Effets secondaires :**
    - Si une condition dans une expression logique contient des fonctions qui modifient l'état d'une variable ou effectuent des actions (exemple : impression, écriture de fichiers), ces actions ne seront pas exécutées si Python n'évalue pas cette partie de l'expression.

```python
def modifier_variable():
    global x
    x = 10
    return True

x = 0
if False and modifier_variable():  # modifier_variable() n'est jamais exécutée
    print("Condition remplie")

print(x)  # Résultat : x reste 0
```

2. **Ordre des conditions :**
    - Placez les conditions les plus simples ou les plus probables en premier pour maximiser les avantages de l'évaluation paresseuse.

En résumé

L'évaluation paresseuse est une optimisation qui :

- Améliore les performances en arrêtant les calculs inutiles.
- Préserve la sécurité dans certains cas (comme la gestion des divisions par zéro).
- Fonctionne naturellement avec les opérateurs logiques `and` et `or`, en respectant leurs comportements.

Ce concept est particulièrement utile pour écrire du code efficace et sécurisé tout en minimisant les erreurs potentielles.