Python - Les opérateurs logiques

Les opérateurs logiques permettent d'évaluer des expressions booléennes et prendre des décisions en fonction de ces évaluations. En Python, les opérateurs logiques les plus couramment utilisés sont `and`, `or`, `xor` et `not`. Dans cet article, nous explorerons chacun de ces opérateurs et leur utilisation dans des situations pratiques.

5GMS 6GMS 3TTR Découverte

💡 Un opérateur logique permet de "combiner" 2 ou plusieurs valeurs booléennes et renvoie une valeur booléenne.

L'opérateur `and`

L'opérateur `and` retourne `True` si **les deux** expressions qu'il relie **sont vraies**, sinon il retourne `False`.

Table de vérité pour l'opérateur `and` :

A	B	A and B
✅ True	✅ True	✅ True
✅ True	❌ False	❌ False
❌ False	✅ True	❌ False
❌ False	❌ False	❌ False

Exemple :

```
1  x = 5
2  y = 10
3  result = (x < 10) and (y > 5)
4  print(result)  # Output: True
```

Dans cet exemple, l'expression `(x < 10) and (y > 5)` est vraie car les deux conditions sont satisfaites : `x` est inférieur à 10 et `y` est supérieur à 5.

Il est bien sûr possible de combiner plus de 2 comparaisons, comme dans l'exemple suivant:

Vérification de l'éligibilité pour un concours

Imaginons qu'on vérifie si une personne est éligible pour participer à un concours. Les critères sont les suivants :

- L'âge doit être supérieur ou égal à 18 ans.
- Elle doit être résidente en Belgique.
- Ca doit être sa première participation au concours.

```python
age = 20
residence = "Belgique"
premiere_participation = True

if age >= 18 and residence == "Belgique" and premiere_participation:
    print("Vous êtes éligible pour participer au concours.")
else:
    print("Vous ne remplissez pas les critères pour participer au concours.")
```

> Il est bien sûr possible de **COMBINER PLUSIEURS COMPARAISONS**. Dans le cas du `and`, l'expression sera considérée comme `True` si **TOUTES les conditions sont** `True`.

L'opérateur `or`

L'opérateur `or` retourne `True` si **au moins l'une** des expressions qu'il relie **est vraie**, sinon il retourne `False`.

Table de vérité pour l'opérateur `or` :

A	B	A or B
✅ True	✅ True	✅ True
✅ True	❌ False	✅ True
❌ False	✅ True	✅ True
❌ False	❌ False	❌ False

Voici un exemple :

```python
x = 5
y = 10
result = (x > 10) or (y < 20)
print(result)  # Output: True
```

Dans cet exemple, l'expression `(x > 10) or (y < 20)` est vraie car au moins l'une des conditions est satisfaites : `y` est inférieur à 20.

> Il est bien sûr possible de **COMBINER PLUSIEURS COMPARAISONS**. Dans le cas du `or`, l'expression sera considérée comme `True` si **au moins UNE des conditions est** `True`.
> Exemple:

```python
x = 5
y = 10
z = 15

if x > 0 and y < 5 and z == 15:
    print("Toutes les conditions sont remplies.")
else:
    print("Au moins une condition est fausse.")

```

L'opérateur `xor`

L'opérateur `xor` (ou exclusif) retourne `True` si une et une seule des expressions qu'il relie est vraie, sinon il retourne `False`. En Python, il n'existe pas d'opérateur `xor` natif, mais vous pouvez le simuler en utilisant l'opérateur `!=` (différent de).

Table de vérité pour l'opérateur `xor` :

A	B	A xor B
✅ True	✅ True	❌ False
✅ True	❌ False	✅ True
❌ False	✅ True	✅ True
❌ False	❌ False	❌ False

Voici un exemple :

```python
x = True
y = False
result = x != y
print(result)  # Output: True
```

Dans cet exemple, l'expression `x != y` est vraie car une des variables est vraie et l'autre est fausse.

L'opérateur `not`

L'opérateur `not` **inverse la valeur** d'une expression booléenne. Si l'expression est vraie, `not` la rendra fausse, et vice versa.

Table de vérité pour l'opérateur `not` :

A	not A
✅ True	❌ False
❌ False	✅ True

Voici un exemple :

```python
x = True
result = not x
print(result)  # Output: False
```

Dans cet exemple, la valeur de `x` est vraie, mais `not x` inverse cette valeur pour retourner `False`.

Ordre de priorité

Lorsque vous utilisez **plusieurs opérateurs logiques** dans une expression Python, il est important de comprendre leur ordre de priorité, également appelé **précédence des opérateurs**. Cela **détermine l'ordre dans lequel les opérateurs sont évalués dans une expression**. Voici l'ordre de priorité des opérateurs logiques, du plus haut au plus bas :

1. **not** : L'opérateur `not` a la plus haute précédence. Il est utilisé pour inverser la valeur d'une expression booléenne.

2. **and** : Ensuite vient l'opérateur `and`. L'expression est évaluée de gauche à droite et l'évaluation s'arrête dès qu'une valeur `False` est trouvée, car la conjonction de deux expressions nécessite que les deux soient vraies.

3. **or** : Enfin, l'opérateur `or` a la plus basse précédence. De la même manière que l'opérateur `and`, l'expression est évaluée de gauche à droite et l'évaluation s'arrête dès qu'une valeur `True` est trouvée, car la disjonction de deux expressions nécessite que au moins une soit vraie.

Il est important de noter que l'utilisation de parenthèses peut modifier l'ordre d'évaluation dans une expression. Les parenthèses ont la priorité la plus élevée et sont évaluées en premier. Par conséquent, vous pouvez utiliser des parenthèses pour spécifier explicitement l'ordre dans lequel les opérateurs doivent être évalués.

En comprenant l'ordre de priorité des opérateurs logiques, vous pouvez écrire des expressions booléennes complexes de manière claire et précise, en vous assurant que l'évaluation se déroule comme prévu.

Conclusion

Les opérateurs logiques en Python peuvent évaluer des expressions booléennes et vous aider à prendre des décisions dans vos programmes. En comprenant comment utiliser `and`, `or`, `xor` et `not`, vous pouvez écrire du code plus expressif et plus concis tout en contrôlant le flux de votre programme en fonction des conditions logiques.

Exercices

Devine le résultat de ces programmes **SANS LES EXÉCUTER** dans Python.

Ensuite, vérifie ton hypothèse en tapant le programme (PAS DE COPIER-COLLER) et en déboguant à l'aide de la touche `F7`.

1. **Exercice 1 :**

```
x = 5
y = 10
result = (x > 3) and (y < 20)
print(result)
```

Devinez le résultat du programme.

2. **Exercice 2 :**

```
a = True
b = False
c = True
result = a and b or c
print(result)
```

Devinez le résultat du programme.

3. **Exercice 3 :**

```
x = 15
y = 25
result = (x < 10) or (y > 20)
print(result)
```

Devinez le résultat du programme.

4. **Exercice 4 :**

```
a = True
b = False
result = not a or b
print(result)
```

Devinez le résultat du programme.

5. **Exercice 5 :**

```
x = 8
y = 12
result = (x % 2 == 0) and (y % 3 == 0)
print(result)
```

Devinez le résultat du programme.

6. Exercice 6 :

```
a = False
b = True
result = a or (not b)
print(result)
```

Devinez le résultat du programme.

7. Exercice 7 :

```
x = 20
y = 30
result = (x > 10) and (y < 25) or (x + y == 50)
print(result)
```

Devinez le résultat du programme.

8. Exercice 8 :

```
a = True
b = True
result = (not a) or (not b)
print(result)
```

Devinez le résultat du programme.

9. Exercice 9 :

```
x = 5
y = 10
z = 15
result = (x + y > 10) and (y - z < 0) or (x * z == 75)
print(result)
```

Devinez le résultat du programme.

10. Exercice 10 :

```
a = True
b = False
result = (a and b) or (not a and not b)
print(result)
```

Devinez le résultat du programme.

Ces exercices vous permettront de tester votre compréhension des opérateurs logiques en Python en essayant de prédire le résultat de chaque expression booléenne.