Opérateurs booléens

1 - Opérateur booléen not

1.1 Tester un prédicat à True

Présentation du prédicat-exemple

Soit le prédicat **contient()** qui permet de tester si une chaine contient un mot précis, en comparant les deux en minuscules.

```
def contient(chaine :str, mot :str) -> bool :
    """Prédicat qui renvoie True si le mot est dans le string"""

    chaine_minus = chaine.lower()  # La chaine mais en miniscules
    mot_minus = mot.lower()        # Le mot mais en miniscules
    return mot_minus in chaine_minus
```

```
>>> contient("Je veux me plaindre de mes notes bla bla", "PLAINDRE")
True
```

Bonne pratique de programmation

On exécute le bloc si l'expression booléenne fournie derrière le **if** est évaluée à **True**. Ce programme affiche les plats qui contiennent des poivrons :

```
plats = [
    "crêpes au sucre : farine de blé ; oeufs ; lait ; sucre",
    "pates carbonara : pâtes(blé) ; fromage ; lardon(porc) ; crême épaisse ; basilic",
    "moules frites : moule ; vin blanc ; oignons ; persil ; frites(pommes de terre)",
    "ratatouille : poivrons verts ; poivrons rouges ; aubergines ; courgettes ; tomates ; oignons ;
    "boeuf aux carottes : boeuf ; carottes ; oignons ; vin blanc ; ail"
    "poulet basquaise : tomates ; poivrons rouges ; poivrons verts ; oignons ; poulet ; thym ; lauri
    ]

def contient(chaine :str, mot :str) -> bool :
    """Prédicat qui renvoie True si le mot est dans le string"""
    chaine_minus = chaine.lower()  # La chaine mais en miniscules
    mot_minus = mot.lower()        # Le mot mais en miniscules
    return mot_minus in chaine_minus

for plat in plats :                  # pour chaque plat du tableau plats
    if contient(plat, "Poivrons") :  # si le plat contient des poivrons
        print(plat)
```

Mauvaise pratique de programmation

Nous aurions pu écrire ceci mais c'est plus long et inutile :

```
for plat in plats :                           # pour chaque plat du tableau plats
    if contient(plat, "Poivrons") == True :   # si le plat contient des poivrons
        print(plat)
```

1.2 Tester un prédicat à False

On veut récupérer les plats qui ne contiennent pas de poivron.

Très mauvaise pratique de programmation

A éviter absolument. De façon naïve, on pourrait se dire : si le prédicat est vrai, on ne fait rien, sinon on agit.

```
for plat in plats :
    if contient(plat, "poivron") :
        pass
    else :
        print(plat)
```

Assez mauvaise pratique de programmation

On peut tester le fait que l'expression soit évaluée à `False`.

```
# pour chaque plat du tableau plats
for plat in plats :
  # si plat ne contient pas poivron
  if contient(plat, "Poivrons") == False :
      print(plat)
```

Bonne pratique de programmation : utiliser not

On peut encore faire mieux en utilisant le mot-clé **`not`** qui inverse la valeur du booléen.

Signature : **`not bool -> bool`**

```
>>> b = False
>>> b
False
```

```
>>> not b
True
```

On peut alors écrire plus proprement le fait d'agir uniquement si le mot **`"Poivron"`** n'apparaît pas :

```
# pour chaque plat du tableau plats
for plat in plats :
    # si plat ne contient pas poivron
    if not contient(plat, "Poivrons") :
        print(plat)
```

2 - Opérateur booléen and

2.1 and pour utiliser plusieurs conditions en même temps

Imaginons qu'on veuille récupérer les plats qui contiennent du poivron **et** de l'aubergine. On dispose du prédicat **`contient()`**.

Méthode possible mais problématique dans certains cas

On filtre sur les poivrons puis sur les aubergines.

```
for plat in plats :
   if contient(plat, "poivron") :
      if contient(plat, "aubergine") :
         print(plat)
```

Si on ne doit rien faire lorsque c'est faux, c'est correct. Par contre, si on doit agir lorsque c'est faux, on doit faire du copier-coller.

```
for plat in plats :
   if contient(plat, "poivron") :
      if contient(plat, "aubergine") :
         print(plat)
      else :
         faire_un_truc()
   else :
      faire_un_truc()
```

Bonne pratique de programmation : utiliser and

On peut faire mieux en utilisant le mot-clé **`and`** : ce mot-clé est un opérateur booléen qui

- **renvoie Vrai uniquement si les entrées sont toutes vraies**
- renvoie Faux dans tous les autres cas.

Signature : **`bool and bool -> bool`**

```
>>> a = True
>>> b = True
>>> a and b
True
```

```
>>> a = True
>>> b = False
>>> a and b
False
```

```
>>> a = False
>>> b = True
>>> a and b
False
```

Exemple d'utilisation : pour afficher les plats qui contiennent poivrons et aubergines :

```python
# pour chaque plat du tableau plats
for plat in plats :
    # si le plat contient poivrons ET aubergines
    if contient(plat, "Poivron") and contient(plat, "Aubergine") :
        print(plat)
```

Notez bien la présence d'une condition à gauche et à droite du mot-clé and.

Si on veut agir lorsque c'est faux, pas de copier-coller :

```python
# pour chaque plat du tableau plats
for plat in plats :
    # si le plat contient poivrons ET aubergines
    if contient(plat, "Poivron") and contient(plat, "Aubergine") :
        print(plat)
    else :  # sinon, il en manque au moins un des deux
        faire_un_truc()
```

2.2 Une erreur typique

Voici une erreur courante que tout le monde va faire un jour ou l'autre.

Imaginons qu'on veuille **vérifier que deux notes soient bien supérieures à 10** : en français, nous dirions que note1 et note2 sont supérieures à 10.

Voici l'erreur sémantique courante :

```python
return note1 and note2 > 10
```

Pourquoi est-ce que cela ne fonctionne pas comme vous le voulez ?

D'abord car **les opérateurs de comparaison sont prioritaires sur les opérateurs logiques**. En réalité, vous avez écrit ceci :

```python
return note1 and (note2 > 10)
```

L'opérateur **and** travaille donc sur les deux propositions suivantes :

— **note1** : faux si **note1** est nulle ou vide, vrai sinon.

— **note2 > 10** : rien d'étrange pour celle-ci.

La traduction qui fait Python de cette ligne est donc : est-ce que **note1** contient quelque chose et est-ce que **note2** est supérieure à 10. Cela n'est donc pas ce que nous voulions.

2.3 La bonne façon de traduire ceci

Il faut parler à Python comme il le désire. Il faut placer une propriété à gauche du **and** et une propriété à droite du **and**.

```python
return note1 > 10 and note2 > 10
```

Les opérateurs de comparaison étant prioritaires sur les opérateurs logiques, c'est comme si nous avions écrit ceci :

```python
return (note1 > 10) and (note2 > 10)
```

Cela veut bien dire ce qu'on voulait au départ : est-ce que **note1** est supérieure à 10 et est-ce que **note2** est supérieure à 10.

3 - Opérateur booléen or

3.1 Principe

Imaginons qu'on veuille récupérer les plats qui contiennent soit du poivron, soit de l'aubergine, voire les deux à la fois. On commence par filtrer sur les poivrons, sinon, on filtre sur les aubergines.

```python
for plat in plats :
    if contient(plat, "poivron") :
        print(plat)
    elif contient(plat, "aubergine") :
        print(plat)
```

Mais on a la même instruction sur les lignes 3 et 5.

3.2 Bonne pratique de programmation : utiliser or

Le mot-clé **not** est un opérateur booléen qui

— renvoie Faux uniquement si les entrées sont toutes fausses

— renvoie Vrai si l'une des entrées est vraie.

Signature : **bool or bool -> bool**

```
>>> a = True
>>> b = True
>>> a or b
True
```

```
>>> a = True
>>> b = False
>>> a or b
True
```

```
>>> a = False
>>> b = True
>>> a or b
True
```

Pour afficher proprement les plats qui contiennent des poivrons OU des aubergines, ou les deux :

```
for plat in plats :                          # pour chaque plat du tableau plats
    if contient(plat, "Poivron") or contient(plat, "Aubergine") :  # si le plat contient poivrons
        print(plat)
```

Notez bien la présence d'une condition à gauche et à droite du mot-clé **or**.

4 - Priorité des opérateurs logiques (P.A.C.Bo)

Par rapport aux autres opérateurs : P.A.C.Bo

P Les plus prioritaires : les parenthèses ()

A Ensuite les opérateurs arithmétiques : `** * / // % + -`

C Ensuite les opérateurs de comparaison : [**in** et **not in**], ensuite [**>**, **>=**...], ensuite [**==**, **!=**]

Bo Ensuite les opérateurs logiques booléens : **not** puis **and** puis **or**

Encore moins prioritaire : **if**, **elif**, **else**

Le moins prioritaire de tous : l'affectation **=**

5 - Tables de vérité

5.1 Table de vérité du non/not

a	not a
False	True
True	False

5.2 Table de vérité du et/and

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

A RETENIR : le ET est VRAI uniquement lorsque toutes les entrées sont à VRAI.

5.3 Table de vérité du ou/or

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

A RETENIR : le OU est FAUX uniquement lorsque toutes les entrées sont à FAUX.

5.4 Table de vérité du ou exclusif/xor

a	b	a xor b
False	False	False
False	True	True
True	False	True
True	True	False

A RETENIR : le **xor** logique n'existe pas nativement en Python, si vous voulez l'utiliser, il faudra utiliser **not(a and b) and (a or b)**.