

# Blocs d'instructions

Polytech Marseille

Séverine Dubuisson, Simon Vilmin

`severine.dubuisson@univ-amu.fr`,

`simon.vilmin@univ-amu.fr`

2024 - 2025

**amU** Aix  
Marseille  
Université



# Au programme

## Structures conditionnelles en Python

- Bloc `if, else`

- Bloc `if, elif, else`

- Résumé

## Structures itératives en Python

- Bloc `while`

- Bloc `for`

- Interruptions avec `break` et `continue`

- Résumé

## À propos des conditions

- Expressions booléennes

- Un peu de logique

- Résumé

# Structures conditionnelles en Python

## Structures conditionnelles en Python

- Bloc `if`, `else`

- Bloc `if`, `elif`, `else`

- Résumé

## Structures itératives en Python

## À propos des conditions

## Bloc `if`, `else`

 **Syntaxe** : pour un *si - sinon* on utilise `if` - `else` en Python

```
if condition:           # si condition est vraie
    instructions si      # instructions indentées
else:                   # sinon (facultatif)
    instructions sinon
```

**i Remarque** : le bloc `else` est *facultatif*, et on ne peut en mettre qu'un seul. Autrement dit, à un `if` est associé *au plus* un `else`

### **! Attention** :

- condition *expression booléenne*  $\Rightarrow$  renvoie `True`, `False` ou `None`
- il faut *indenter* les instructions dans le bloc
- les lignes avec `if` et `else` se terminent par « : »
- pas de condition après le `else`

## Exemples

```
mot = input("dites oui : ")
if mot == "oui":
    print(f"{mot}stiti !")
```

```
# ==== Resultat
dites oui : non
dites oui : oui
ouistiti !
```

```
age = int(input("entrez votre age : "))
if age < 18:
    print(f"vous avez {age} ans, vous etes donc mineur.e !")
else:
    print(f"vous avez {age} ans, vous etes donc majeur.e !")
```

```
# ==== Resultat
entrez votre age : 207
vous avez 207 ans, vous etes donc majeur.e !
```

## Exercice

⚙️ **Exercice** : écrire un programme qui saisit un nombre réel  $x$ , et qui affiche «  $x$  est plus grand que 0 » si  $x$  est plus grand que 0 et «  $x$  est négatif ou nul » sinon (remplacer  $x$  par sa valeur dans les messages).

⚙️ **Exercice** : refaire l'exercice précédent mais sans utiliser de bloc `if`, `else`.



### Astuce :

```
>>> int(True)
1
>>> int(False)
0
>>> x = 4
>>> 2 * f" {x} !"
4 ! 4 !
```

⚙️ **Exercice** : écrire un programme qui saisit trois nombre entiers  $a$ ,  $b$ ,  $c$  et qui affiche le plus grand des trois.

## À propos de l'utilisation de `if`



**Astuce :** si les traitements effectués après le `if` et le `else` sont similaires, probablement pas besoin d'en faire un bloc !

```
n = 25
if (n > 0):      # si la condition est vraie
    print(True) # afficher vrai
else:           # sinon (si la condition est fausse)
    print(False) # afficher faux
```



**Remarque :** ici, aucune différence entre le `if` et le `else`. Dans les deux cas, on affiche le résultat du test `n > 0`

```
n = 25
print(n > 0) # 'factorisation'
```

## Interlude avant la suite

Un programme Python est souvent divisé en deux parties :

- des fonctions ou des classes utiles pour le programme
- le programme principal

Le programme principal se situe souvent à l'intérieur d'une condition :

```
if __name__ == "__main__":  
    instructions # indentation !
```

En bref, si le programme est appelé par un autre, le code du `if` n'est pas exécuté

### **Remarque** : conséquences pour nous

- permet de comprendre cette condition dans des programmes existants
- l'utiliser pour respecter les standards
- implicite dans les slides par soucis de place !



## Conditions imbriquées

**! Attention :** il est possible d'imbriquer les conditions, mais il faut alors *imbriquer les indentations* !

```
if condition1:
    # 1er niveau d'indentation
    if condition2:
        # 2eme niveau d'indentation
    else:
        # 2eme niveau d'indentation
        if condition3:
            # 3eme niveau d'indentation
...

```



**Astuce :** nombre d'indentation = profondeur des blocs imbriqués

# Exemple 1

```
temp = float(input("entrez la temperature ambiante : "))
if 27.2 < temp < 42:
    heure = int(input("entrez l'heure (arrondie) : "))
    if heure > 21:
        print("c'est la canicule !")
    else:
        print("il fait (tres) chaud mais ca va peut-etre refroidir la nuit")
else:
    if temp > 42 or temp < -10:
        print("billevesees !")
    else:
        print("il fait a peu pres bon ca va")

# ==== Resultat
entrez la temperature ambiante : 28
entrez l'heure (arrondie) : 10
il fait (tres) chaud mais ca va peut-etre refroidir la nuit
entrez la temperature ambiante : 77
billevesees !
```

## Exemple 2

```
temps = input("quel temps fait-il ? ")
if temps == "pluie":
    print("s'il y a pluie, il y a parapluie")
else:
    if temps == "beau":
        print("s'il fait beau, c'est chapeau")
    else:
        if temps == "nuageux":
            print("si le temps est nuageux, c'est pas tres lumineux")
        else:
            print("temps etrange, je mange une orange")

# ==== Resultat
quel temps fait-il ? beau
s'il fait beau, c'est chapeau
quel temps fait-il ? dofjghqep
temps etrange, je mange une orange
```

## Exercice

⚙️ **Exercice** : le but est d'écrire un programme qui « filtre » des mails. Le programme demande la saisie d'un type de mail et

- si le mail est un spam, le programme affiche « spam supprime »
- sinon, si le mail est une notification de réseau social, le programme affiche « message dans l'onglet forums »
- sinon, si le mail est une promotion, le programme affiche « go promos »
- sinon, le programme affiche « nouveau message dans la boîte principale »

⚙️ **Exercice** : écrire un programme qui résout une l'équation  $ax + bx^2 + c = 0$  dans  $\mathbb{R}$ . Les paramètres  $a$ ,  $b$ ,  $c$  sont demandés via `input()`.

## Bloc `if`, `elif`, `else`

❌ **Problème** : pas toujours très pratique d'imbriquer des conditions ...

📖 **Syntaxe** : pour modéliser un *si - sinon si - sinon*, on peut utiliser `elif` :

```
if condition1:
    instructions si
elif condition2:
    instructions sinon si
else:
    instructions sinon
```

### 📌 **Remarque** :

- permet d'éviter d'imbriquer certains `if`, `else`
- autant de `elif` que l'on veut
- toujours penser à l'indentation et au « : »
- le `else` toujours en dernier et facultatif

## Exemple 1

```
temps = input("quel temps fait-il ? ")
if temps == "pluie":
    print("s'il y a pluie, il y a parapluie")
elif temps == "beau":
    print("s'il fait beau, c'est chapeau")
elif temps == "nuageux":
    print("si le temps est nuageux, c'est pas tres lumineux")
else:
    print("temps etrange, je mange une orange")
```

```
# ==== Resultat
quel temps fait-il ? beau
s'il fait beau, c'est chapeau
quel temps fait-il ? dofjghqep
temps etrange, je mange une orange
```

## Exemple 2

```
temp = float(input("entrez la temperature ambiante : "))
if 27.2 < temp < 42:
    heure = int(input("entrez l'heure (arrondie) : "))
    if heure > 21:
        print("c'est la canicule !")
    else:
        print("il fait (tres) chaud mais ca va peut-etre refroidir la nuit")
elif temp > 42 or temp < -10:
    print("billevesees !")
else:
    print("il fait a peu pres bon ca va")

# ==== Resultat
entrez la temperature ambiante : 28
entrez l'heure (arrondie) : 10
il fait (tres) chaud mais ca va peut-etre refroidir la nuit
entrez la temperature ambiante : 77
billevesees !
```

# Exercice

⚙️ **Exercice** : reprendre les exercices précédents en remplaçant, si possible, les `if`, `else` imbriqués par des `if`, `elif`, `else`.



# Résumé



## Rappel :

- pour un bloc conditionnel, on utilise `if`, `elif`, `else`
- peuvent être imbriqués
- pour un bloc : au plus un `else`, autant de `elif` que l'on veut



**Attention :** veiller à *indenter* et mettre « : » après `if`, `elif`, `else`

```
n = int(input("entrez un entier positif <= 9 : "))
if (n < 0) or (n > 9):
    print(f"{n} est en dehors de l'intervalle !")
else:
    if n == 6:
        print("nombre parfait !")
    elif (n > 1) and ((n % 2 != 0) or (n % 3 != 0)):
        print(f"{n} est un nombre premier")
    else:
        print(f"{n} n'est ni premier, ni parfait")
```

# Structures itératives en Python

## Structures conditionnelles en Python

### Structures itératives en Python

- Bloc `while`


- Bloc `for`

- Interruptions avec `break` et `continue`

- Résumé

### À propos des conditions

## Bloc `while`

 **Syntaxe** : pour exprimer une boucle *tant que*, qui se répète jusqu'à ce qu'une condition change, on utilise `while`

```
while condition: # tant que condition est vraie
    instructions # instructions (indentées)

suite_programme
```

### Remarque :

- le contenu de la boucle est répété tant que condition est *vraie*
- une fois la boucle terminée, on passe à `suite_programme`
- comme pour `if`, `else`, il faut *indenter* les instructions dans la boucle
- comme pour `if`, `else`, il faut mettre « : » après la condition du `while`

## Exemple 1

```
action = "continue"

while action != "stop":
    action = input("prochaine action : ")
    print(f"on effectue {action}")

print("fini !")

# ==== Resultat
prochaine action : pouet
on effectue pouet
prochaine action : NON
on effectue NON
prochaine action : stop
fini !
```

## Exemple 2

```
n = int(input("entrez un entier n a decomposer : "))
i = int(input(f"entrez un entier i par lequel decomposer {n} : "))
quotient = 0 # quotient de n / i
reste = n # reste

while reste > i:
    quotient = quotient + 1
    reste = reste - i

print(f"on a ({i} x {quotient}) + {reste} = {n}")

# ==== Resultat
entrez un entier n a decomposer : 111
entrez un entier i par lequel decomposer 111 : 7
on a 7 x 15 + 6 = 111
```

## Exemple 3

```
stop = False
action = ""

while not(stop):
    titre = input("titre de film : ")
    note = int(input("note sur 5 : "))

    while (note < 0) or (note > 5):
        note = int(input("sur 5 on a dit : "))

    print(f"{titre} ==== {note}/5")
    action = input("continue ou stop ? ")
    stop = (action == "stop")
    print("\n")

print("Fin")
```

```
# ==== Resultat
titre de film : Solaris
note sur 5 : 4
Solaris ==== 4/5
continue ou stop ? continue

titre de film : Sucker Punch
note sur 5 : -1000
sur 5 on a dit : 0
Sucker Punch ==== 0/5
continue ou stop ? continue

titre de film : Dante 01
note sur 5 : 2
Dante 01 ==== 2/5
continue ou stop ? stop

Fin
```

## Boucles infinies

✘ **Problème** : exécutons le programme de la division avec  $n = -7$  et  $i = -8$  ... on en sort jamais !

! **Attention** : les *boucles infinies* arrivent si la condition de la boucle n'est jamais fausse

```
n = 4
while n % 2 == 0: # tant que n est pair
    n = 2 * n
    print(n)

print("Fin")
```

# Compteurs, accumulateurs, drapeaux

Quelques types de variables que l'on rencontre dans les boucles :

- *compteurs* : comptent une quantité qui dépend du nombre d'itérations
- *accumulateurs* : stockent de l'information au fur et à mesure de la boucle
- *drapeaux* : booléens relatifs aux valeurs parcourues, à un compteur ou un accumulateur

Exemples :

- *accumulateurs* : somme de valeurs, produit, nombre de voyelles, ...
- *drapeaux* : seuil sur le nb d'itérations, présence d'un élément ...

**i Remarque :** bien penser à initialiser les variables accumulateurs, compteurs, drapeaux *avant la boucle*



## Exemple

```
n = -1
n_valide = False          # drapeau sur la validite de n
nb_iteration = 0          # compteur d'iterations
saisie_bloquee = False    # drapeau sur le nb de saisies

while not(n_valide) and not(saisie_bloquee):
    n = int(input("entrez un entier positif :"))
    nb_iteration = nb_iteration + 1 # mise a jour du compteur
    n_valide = (n >= 0)             # mise a jour du drapeau
    saisie_bloquee = (nb_iteration >= 5)

if saisie_bloquee:
    print("votre saisie est bloquee, au revoir")
else:
    somme = 0 # accumulateur de la somme des entiers
    reste = n
    while reste > 0:
        somme = somme + reste # mise a jour de l'accumulateur
        reste = reste - 1    # reste joue le role de compteur
    print(f"la somme des {n} premiers entiers est {somme}")
```

## Exercice

⚙️ **Exercice :** (*le nombre mystère*) On cherche à faire deviner un nombre choisi au hasard à l'utilisateur·ice. Le programme

- choisit un nombre `alea` au hasard entre 0 et 1000
- demande à l'utilisateur·ice de saisir un nombre `choix` et recommence tant que `alea` n'a pas été trouvé
- affiche le message « trop grand » ou « trop petit » en fonction de si `choix < alea` ou non
- affiche à la fin le nombre d'essais réalisés




**Astuce :** pour choisir un nombre aléatoirement

```
# on importe la fonction randint du module random
from random import randint

# choix d'un nombre entre 0 et 1000
alea = randint(0, 1000)
```

## Bloc for


 **Syntaxe** : quand on connaît le nombre d'itérations à faire dans une boucle, on peut utiliser `for` et `range()`

```
# pour i allant de start a end de step en step
for var in range(start=0, end, step=1):
    instructions

suite_programme
```

### Remarque :

- comme toujours, *indentation* et « : »
- on verra plus tard que le `for` sert aussi à parcourir des types structurés
- dans le `range`, `start` et `step` sont facultatifs (0 et 1) par défaut

 **Attention** : l'itération suit la formule `start + step * i` tant que `end` n'est pas atteint/dépassé. L'élément `i` commence à 0.

## Exemple 1

```
for i in range(5): # start = 0, end = 5, step = 1
    print(i, end=" ")
```

```
# ==== Resultat
```

```
0 1 2 3 4
```

```
# pas de 5 car (deb + i * pas) >= 5 quand i = 5 !
```

```
for i in range(5, 1, -1): # de 5 a 1 par step (par pas) de -1
    print(i, end=" ")
```

```
# ==== Resultat
```

```
5 4 3 2
```

## Exemple 2

```
for i in range(1, 5, -1): # de 1 a 5 par step de -1
    print(i, end=" ")

# ==== Resultat

# range s'arrange en interne pour eviter les boucles infinies !

n = int(input("entrez un entier positif :"))
somme = 0
for i in range(n + 1):
    somme = somme + i
print(f"la somme des {n} premiers entiers est {somme}")

# ==== Resultat
entrez un entier positif : 5
la somme des 5 premiers entiers est 15
```

## Exemple 3

```
n = int(input("entrez un entier positif :"))
for i in range(n):
    print(i * " ", end = "")
    for j in range(n - i):
        print(i, end=" ")
    print() # pour retourner a la ligne
```

```
# ==== Resultat
```

```
entrez un entier positif : 5
```

```
0 0 0 0 0
```

```
 1 1 1 1
```

```
  2 2 2
```

```
   3 3
```

```
    4
```

## Exercice

⚙️ **Exercice** : écrire un programme qui lit un mot au clavier et qui vérifie qu'il contient autant de 'a' que de 'b'.

⚙️ **Exercice** : écrire un programme qui lit deux entiers  $n$ ,  $m$  au clavier et qui affiche une matrice de dimension  $n \times m$  dans laquelle le coefficient  $a_{ij}$  est le nombre obtenu par concaténation de  $i$  et  $j$ .

🐾 **Astuce** : par exemple, pour  $n = 2$  et  $m = 3$ , le programme affiche :

```
11 12 13
21 22 23
```

⚙️ **Exercice** : que choisir ? `for` ou `while` :

- la moyenne de  $n$  nombres saisis au clavier,  $n$  est également saisi au clavier
- calculer le prix total d'un ensemble d'articles dont le prix est lu au fur et à mesure

Proposer des programmes pour ces deux cas.

# Itération sur les chaînes de caractères



**Rappel :** dans une chaîne de caractères, on peut accéder à chaque caractère par leur indice.

Les chaînes de caractères sont *itérables* : on peut les *parcourir*.

```
chaîne = "bonjour"
for i in chaîne: # pour chaque caractere de la chaîne
    print(i, end=" ")

# ==== Resultat
b o n j o u r
```



## Le mot-clé `break`

 **Syntaxe** : on peut *arrêter* l'exécution d'une boucle avec le mot clé `break`

```
while condition1:  
    instructions  
    break  
    instructions
```

```
for iteration:  
    instructions  
    break  
    instructions
```

**i Remarque** : quand le `break` est atteint, on sort purement et simplement de la boucle. Le mot-clé `break` n'est utilisable *que* dans les boucles.

## Exemple

```
for i in range(1, 10):  
    print(i, end=" ")  
    if i == 3: # si i vaut 3  
        break # on sort
```

```
# ==== Resultat
```

```
0 1 2
```

```
while True:  
    # boucle infinie  
    print("AAAAH")  
    break
```

```
print("Fin")
```

```
# ==== Resultat
```

```
AAAAH
```


```
Fin
```

## Exemple

```
prenom = input("Saisir un prenom : ")
iter = 0
while iter < 10:
    print(f"Bonjour {prenom}, ca va ?")
    prenom = input("saisir un prenom : ")
    if prenom == "Jacques":
        print(f"ah nan ! pas {prenom}")
        break
```


```
# ==== Resultat
Saisir un prenom : Madeleine
Bonjour Madeleine, ca va ?
Saisir un prenom : Nada
Bonjour Nada, ca va ?
Saisir un prenom : Jacques
ah nan ! pas Jacques
```

## Le mot-clé `continue`

 **Syntaxe** : on peut *arrêter* l'exécution de l'*itération courante* avec le mot clé `continue`

```
while condition1:  
    instructions  
    continue  
    instructions
```

```
for iteration:  
    instructions  
    continue  
    instructions
```

 **Remarque** : quand le `continue` est atteint, on passe à la prochaine itération de la boucle sans exécuter la suite du code. À l'instar de `break`, le mot-clé `continue` n'est utilisable *que* dans les boucles.

## Exemple

```
for i in range(5):  
    if i == 3:  
        print("tut", end=" ")  
        continue  
    print(i, end=" ")
```

```
# ==== Resultat
```

```
0 1 2 tut 4
```


```
action = "continue"  
while action != "stop":  
    action = input("prochaine action : ")  
    if action != "continue":  
        continue  
    print(f"bon, on {action}")  
print("enfin !")
```

```
# ==== Resultat
```

```
prochaine action : pouet  
prochaine action : continue  
bon, on continue  
prochaine action : stop  
enfin !
```

## Que choisir ?

 **Question :** drapeaux, `break`, `continue`, comment choisir ?

 **Réponse :** Il n'y a pas de règles absolues, mais on veut un code *clair* et *lisible*. Donc, on essaye de produire un code qui soit *aussi proche que possible* de l'algorithme en *langage naturel*.

## Exemple

« saisir des entiers *jusqu'à* l'obtention d'un nombre pair »

- on ne connaît pas le nombre d'itérations
- on continue tant que le nombre que l'on a saisi n'est pas pair

```
n, pair = -1, False
while not(pair):
    n = int(input("entrez un entier pair :"))
    pair = (n % 2 == 0)
```

« parcourir un mot *jusqu'à* trouver la lettre 'u' »

- on connaît le nombre d'itérations
- on le parcourt et on s'arrête dès qu'on trouve la lettre 'u'

```
for c in "pantoufle":
    print(c, sep=" ")
    if c == "u":
        break
```

## Exercice

⚙️ **Exercice** : écrire un programme qui saisit un mot au clavier et qui vérifie qu'il contient autant de 'a' que de 'b'. Si le mot contient une autre lettre, le programme s'arrête.

⚙️ **Exercice** : écrire un programme qui demande à l'utilisateur·ice de saisir des noms de personne et leur statut à AMU. Si le statut est « prof·e », le programme affiche juste « ok » et passe à la prochaine personne. Si le statut est « étudiant·e » le programme procède à la saisie de 5 notes au clavier et s'arrête à la première note en dessous de 6 (s'il y en a une) et affiche le message « aïe ». À la fin de la saisie des notes, la moyenne est affichée, elle vaudra 0 si il y a eu une note en dessous de 6. Le programme doit intégrer un moyen d'arrêter la saisie.



# Résumé

## Rappel :

- pour coder des boucles pour et tant que : `for` et `while`
- on connaît le nombre d'itérations ? oui `for`, non `while`
- on peut utiliser les mots-clés `break` et `continue` pour arrêter la boucle ou passer à l'itération suivante

 **Attention :** éviter les *boucles infinies*

```
n = 0
while n >= 0:
    n = int(input("entrez un entier positif :"))
    for i in range(n - 1, 1, -1):
        if n % i == 0:
            print(f"{i} divise {n}")
            break
```

# À propos des conditions

Structures conditionnelles en Python

Structures itératives en Python

À propos des conditions

- Expressions booléennes

- Un peu de logique

- Résumé

**i Remarque :** les détails techniques de cette partie ne seront *pas à l'examen*. L'objectif c'est de *comprendre l'importance de réfléchir* à ce qu'on fait dans un programme.


## Objectif de cette partie

```
n = int(input("entrez un nombre :"))
if n < 0:
    if n % 2 == 0:
        print("pair")
    else:
        print("impair")
elif n >= 0 :
```


```
    if n % 2 == 0:
        print("pair")
    else:
        print("impair")
```

```
n = int(input("entrez un entier :"))
if (n < 0) and (n % 2 == 0):
    print("pair")
elif (n >= 0) and (n % 2 == 0):
    print("pair")
else:
    print("impair")
```

```
n = int(input("entrez un entier :"))
if n % 2 == 0:
    print("pair")
else:
    print("impair")
```

 **Idée** : ces trois programmes sont équivalents, mais en *repensant les conditions*, on améliore la *qualité* et la *lisibilité* du code.

# Expressions

 **Définition :** une *expression* est une formule/instruction qui peut être évaluée et qui renvoie une valeur.


```
>>> (6 * 7 + 4) == True # expression
```

```
False
```

```
>>> len("bonjour") # expression
```

```
7
```

```
>>> a = 4 # pas une expression
```

 **Remarque :** généralement, une expression (bien formée) est composée de constantes, de variables, d'opérateurs, d'appels de fonctions et respecte la syntaxe de ces éléments.

# Expressions booléennes

Les conditions dans les `while`, `if`, etc, sont des *expressions booléennes*.


 **Définition** : une expr. est *booléenne* si elle renvoie `True` ou `False`

On peut créer des expressions booléennes simples avec :

- les constantes `True` et `False`
- des comparaisons, par ex `4 <= 7` (qui renvoie `False`)
- des opérateurs logiques appliqués à des expressions basiques, par ex `not(4 <= 7) and (4 % 2 == 0)` (qui renvoie `True`)

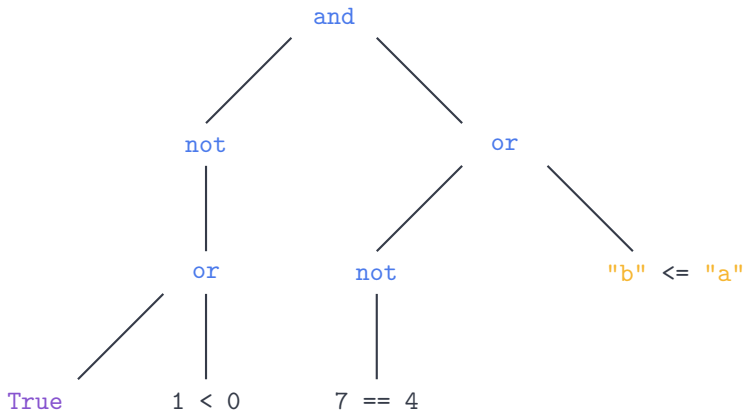
On peut aussi combiner des expressions booléennes pour en créer des nouvelles :

- si `e1` est une expr. bool., `not(e1)` en est une aussi
- si `e1` et `e2` sont des expr. bool., alors `e1 or e2` et `e1 and e2` en sont aussi

 **Remarque** : les formules logiques sont construites de manière *inductive*

## Exemple et principe d'arbre de décomposition

```
not(True or (1 < 0)) and (not(7 == 4) or ("b" < "a"))
```



## Tables de vérités

Les opérateurs `or`, `and`, `not` se comportent de manière différente ;

- `not(e1)` est `True` exactement quand `e1` est `False`
- `(e1 or e2)` est `True` si *au moins l'une* des deux expressions `e1`, `e2` l'est
- `(e1 and e2)` donne `True` exactement quand `e1` *et* `e2` le sont

e1	<code>not(e1)</code>
False	True
True	False

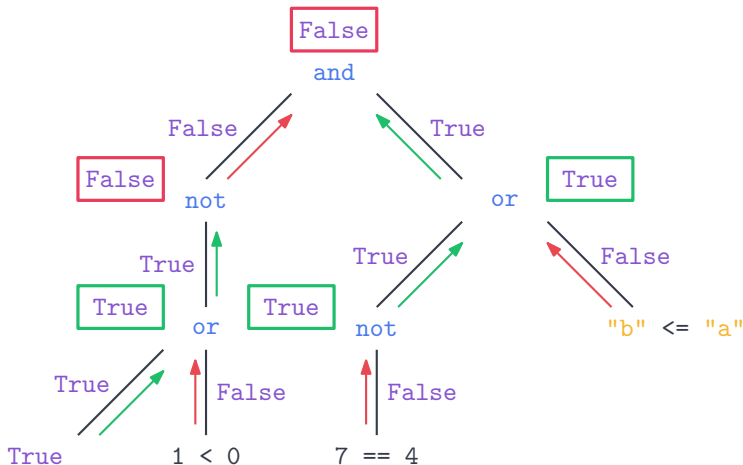
e1	e2	<code>e1 or e2</code>
False	False	False
False	True	True
True	False	True
True	True	True

e1	e2	<code>e1 and e2</code>
False	False	False
False	True	False
True	False	False
True	True	True

**i Remarque :** ces tableaux sont des *tables de vérités* : on associe à chaque combinaisons de valeurs de vérité la valeur résultat.


## Exemple

```
not(True or (1 < 0)) and (not(7 == 4) or ("b" < "a"))
```





## Quelques règles de calcul logique

 **Définition :** Deux expressions logiques (booléennes) sont *équivalentes* si elles ont les mêmes tables de vérités.

On peut *raisonner* sur des expressions grâce à des règles de logique telles que :

- la distributivité

```
e1 or (e2 and e3) == (e1 or e2) and (e1 or e3)
e1 and (e2 or e3) == (e1 and e2) or (e1 and e3)
```

- les lois de De Morgan

```
not(e1 and e2) == not(e1) or not(e2)
not(e1 or e2) == not(e1) and not(e2)
```

- la double négation `not(not(e1)) == e1`

 **Remarque :** règles de *logique propositionnelle* vraies au delà du Python

# Python le fainéant

Python évalue les expressions de manière *feignante*, de *gauche à droite*

- dans `e1 or e2`, il va renvoyer `True` si `e1` est `True` et *ne pas évaluer* `e2`
- dans `e1 and e2`, il va renvoyer `False` si `e1` est `False` *ne pas évaluer* `e2`

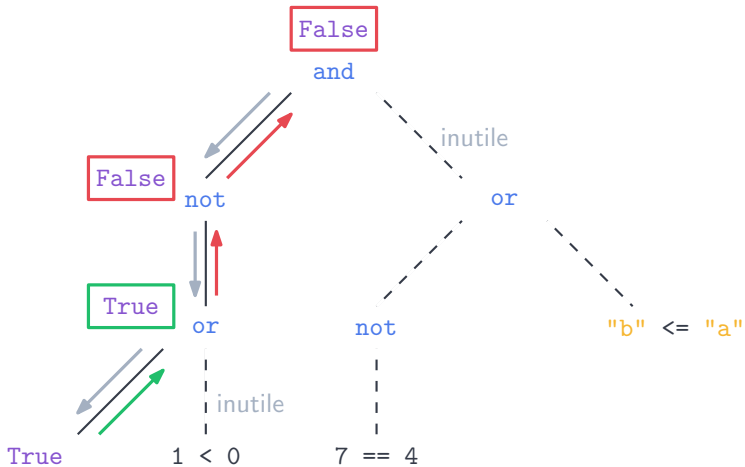
**! Attention :** dans `e1 or e2` (par ex), on peut imaginer que `e2` est mal définie (par ex en utilisant des avec des variables non déclarées), mais que l'évaluation fonctionne quand même. *C'est à éviter!*

```
b = 4
if (b <= 4) or (a = 7): # a pas defini !
    print("pouet")

# == Resultat
pouet
```

## Exemple

```
not(True or (1 < 0)) and (not(7 == 4) or ("b" < "a"))
```



# Résumé



**Rappel :** on peut améliorer un programme en reformulant les conditions qui s'y trouve

- en réfléchissant à leur sens, par ex «  $n$  est pair » est équivalent à «  $n = 2$ , ou  $n = 4$ , ou ... » mais l'une des deux conditions est mieux que l'autre ...
- en utilisant la logique et les tables de vérités pour les simplifier
- en ordonnant les membres de l'expression pour tirer partie de la fainéantise de Python



**Remarque :** *pas besoin d'apprendre en détails* ce qu'on a vu dans cette partie. Le but c'est de *comprendre qu'il faut réfléchir* à ce qu'on fait quand on programme et qu'il existe des outils pour nous aider.