Définition

La **conditionnelle** est un **pilier de la programmation**. Elle va nous permettre de tester une affirmation ou une expression et de va retourner `True` ou `False`. C'est dans les conditions que nous utilisons les les opérateurs de comparaison.

La condition permet de **créer des branches** dans lequel nous écrivons du code. Le code sera alors exécuté seulement si la condition attachée à cette branche a été évaluée comme `True`.

1. Une branche doit adopter la syntaxe suivante : `if | elif + condition + ':'`.

```
condition = ((2 + 2) == 4)
if condition: # condition jugée vraie
    print("2 + 2 est bien égal à 4.") # code de la branche éxécuté
```

2. On peut tout à fait enchaîner les conditions ou les imbriquer.

```
pi = 3.14
if pi > 1:
    print("pi est bien supérieur à 1")
    if pi > 2:
        print("pi est bien supérieur à 2 aussi !")
```

3. Dans le cadre d'une condition `if.. else` seule une des deux branches sera évaluée quoiqu'il arrive.

```
age = 17
if age >= 18:       # sera évaluée comme False
    print("majeur")
else:
    print("mineur") # le code sera exécuté sans même évaluer la condition
```

4. Dans une condition `if.. elif.. else` plusieurs conditions peuvent être évaluées jusqu'à ce qu'on trouve une condition qui renvoie `True`.

```
age = 18
if age > 18:                    # sera évaluée False
    print("vétéran")
elif age == 18:                 # sera évaluée True
    print("tout juste majeur")
else:                           # ne sera jamais évaluée
    print("mineur")
```

Importance de l'indentation

En python, **l'indentation est extrêmement importante** et c'est ce qui remplace les accolades dans la plupart des autres langages vis à vis des conditions. Il est donc important que le code situé dans une des branches de la condition **soit décalé de 1 tab** par rapport la condition elle-même.

```
age = 17
if age >= 18:
print("majeur") # ce code soulèvera une erreur IndentationError
```

Opérateur ternaire

Vous pouvez écrire une condition **sur une seule ligne** grâce à l'opérateur ternaire. À utiliser avec mesure et discernement pour garder un code clair.

```
age = 17
print("majeur") if age >= 18 else print("mineur")
```

Autre exemple :

```
x = 10
y = 20

# Using the ternary operator to assign the smaller value to 'min_value'
min_value = x if x < y else y

print(min_value) # Output: 10
```

Les opérateurs logiques

Les opérateurs logiques permettent de faire des conditions plus complexes.

opérateur	signification
`and`	ET / AND
`or`	OU / OR
`^`	OU EXCLUSIF / XOR
`not`	PAS / NOT

```
nom = "Gérard"

cond_and = nom.startswith('G') and nom.endswith('d')         # ET
cond_or  = nom.startswith('G') or nom.endswith('d')          # OU
cond_xor = nom.startswith('G') ^ nom.endswith('d')           # OU EXCLUSIF
cond_not = not nom.startswith('x') and not nom.endswith('d') # PAS
cond_in  = not 'r' in nom or not 'x' in nom                  # DANS

print(cond_and) # True car "Gérard" commence bien par 'G' et fini bien pars 'd'
print(cond_or)  # True aussi car au moins une des conditions est vraie
print(cond_xor) # False car une seule des conditions doit être vraie
print(cond_not) # False car "Gérard" ne commence pas par 'x' mais fini par 'd'
print(cond_in)  # True car certes, r est dans "Gérard" mais pas 'x'
```

Matrice des opérateurs logiques

A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Le match case

Nouveauté apportée depuis la 3.10, Python propose un **match case** dont la syntaxe rappelle le switch case présent dans la plupart des langages de programmation. Il rempli également la même fonction, mais **apporte de puissantes fonctionnalités supplémentaires**.

Égalité de valeur

Le match case est particulièrement pertinent lorsqu'on sait d'avance quelles **valeurs déterminées** une variable va prendre dans le futur et que l'on doit organiser les conditions en conséquence.

```
mot = "toto"

match mot:
    case "titi":
        print("titi")
    case "tata":
        print("tata")
    case "toto":
        print("toto")
```

Égalité de motif

La fonctionnalité supplémentaire qu'apporte le match case, est qu'il est prévu pour fonctionner avec des **égalités de motifs**.

```
coords = [(1, 3, 7), (34, 2), (123, 2, 82)]

for coord in coords:
    match coord:
        case [x, y]:
            print("Coordonnées en 2 dimensions")
        case [x, y, z]:
            print("Coordonnées en 3 dimensions")
```

On peut même rajouter des conditions à l'intérieur des `case` pour rajouter de la précision.

```
coords = [(0, 0), (1, 3, 7), (34, 2), (123, 2, 82), (0, 0, 0)]

for coord in coords:
    match coord:
        case [x, y]:
            print("Coordonnées en 2 dimensions")
        case [x, y] if x == y == 0:
            print("Coordonnées en 2 dimensions situées à l'origine du repère")
        case [x, y, z] if x == y == z == 0:
            print("Coordonnées en 3 dimensions situées à l'origin du repère")
        case [x, y, z]:
            print("Coordonnées en 3 dimensions à des endroits aléatoires")
```

> 📌 Synthaxe du match case
>
> On remarquera que contrairement aux autres languages de programmation dans lequel le match case est implémenté, ici le mot `break` n'est pas obligatoire entre les différents cases.

Exercice: "11. Les structures conditionnelles"