Les structures conditionnelles

Résumé

Un programme qui exécute toujours les mêmes instructions dans le même ordre est très limité. Pour devenir utile, il doit pouvoir **prendre des décisions** : si l'utilisateur est majeur, afficher un message ; sinon, en afficher un autre. Les **structures conditionnelles** (`if`, `elif`, `else`) permettent au programme de choisir quelle portion de code exécuter selon qu'une condition est vraie ou fausse. C'est la brique fondamentale qui rend un programme intelligent : sans elles, aucun jeu, aucune application, aucun algorithme ne pourrait fonctionner.

Prérequis : Les types de base.

① Mémo

Définition 1 (La structure `if / elif / else`)

```
if condition1:
    bloc1          # exécuté si condition1 est vraie
elif condition2:
    bloc2          # exécuté si condition1 est fausse ET condition2 est
   vraie
else:
    bloc3          # exécuté si toutes les conditions sont fausses
```

- Le `elif` et le `else` sont facultatifs.
- On peut enchaîner autant de `elif` que nécessaire.
- L'indentation (quatre espaces) délimite les blocs.

Opérateurs de comparaison

`==` Égalité (ne pas confondre avec = qui est l'affectation).
`!=` Différent de.
`<`, `<=`, `>`, `>=` Comparaisons d'ordre.
`in` Appartenance : `"a" in "chat"` renvoie `True`.

Opérateurs logiques

`and` Conjonction : `True` si les deux opérandes sont vrais.
`or` Disjonction : `True` si au moins un opérande est vrai.

`not` Négation : inverse la valeur de vérité.
Priorité : `not` > `and` > `or`. En cas de doute, utiliser des parenthèses.

⚠ Pièges fréquents

- Écrire `if x = 5` au lieu de `if x == 5` (affectation au lieu de comparaison).
- Écrire `if x == True` au lieu de simplement `if x`.
- Oublier les deux-points après la condition.
- Tester un intervalle avec `if 0 < x and x < 10` alors que Python autorise l'écriture chaînée : `if 0 < x < 10`.

⚠ Erreurs classiques

Code erroné	Code correct	Explication
`if x = 5:`	`if x == 5:`	= est l'affectation, == est la comparaison : Python lève une `SyntaxError`.
`if x > 0 and < 10:`	`if 0 < x < 10:`	Chaque côté de `and` doit être une expression complète, ou utiliser le chaînage Python.
`not x > 5 and x < 10`	`not (x > 5 and x < 10)`	`not` s'applique seulement à `x > 5` : il faut des parenthèses pour nier toute la condition.
`if x:` quand `x = 0`	Vérifier que `0`, `""`, `[]`, `None` sont `False`	En Python, ces valeurs sont « fausses » : un test implicite peut donner un résultat inattendu.

② Exemples

Programme 1 · Classification d'un nombre

```
def signe(n):
    if n > 0:
        return "positif"
    elif n < 0:
        return "négatif"
    else:
        return "nul"
```

Programme 2 · Année bissextile

```python
def est_bissextile(annee):
    # Divisible par 4 MAIS pas par 100, SAUF si divisible par 400
    return (annee % 4 == 0 and annee % 100 != 0) or (annee % 400 == 0)
```

Programme 3 · Mention au baccalauréat

```python
def mention(moyenne):
    if moyenne >= 16:
        return "Très bien"
    elif moyenne >= 14:
        return "Bien"
    elif moyenne >= 12:
        return "Assez bien"
    elif moyenne >= 10:
        return "Admis"
    else:
        return "Non admis"
```

Programme 4 · Évaluation paresseuse (short-circuit)

```python
# Python n'évalue pas la seconde condition si la première suffit
def divise(a, b):
    if b != 0 and a % b == 0:   # pas d'erreur si b == 0
        return True
    return False
```

Programme 5 · Comparaisons chaînées

```python
x = 7
print(0 < x < 10)     # True (équivaut à 0 < x and x < 10)
print(1 <= x <= 5)     # False
```

③ Exercices

Exercice 1 *Catégorie d'âge* Écrire une fonction `categorie(age)` qui renvoie :

- `"enfant"` si l'âge est strictement inférieur à 12 ;
- `"adolescent"` entre 12 et 17 inclus ;
- `"adulte"` à partir de 18.

Exercice 2 *Triangle valide* Écrire une fonction `type_triangle(a, b, c)` qui prend trois longueurs et renvoie :

- `"invalide"` si l'inégalité triangulaire n'est pas vérifiée ;
- `"équilatéral"` si les trois côtés sont égaux ;
- `"isocèle"` si exactement deux côtés sont égaux ;
- `"quelconque"` sinon.

Exercice 3 *Opérateurs logiques* Sans exécuter le code, déterminer la valeur affichée pour chaque ligne :

```
a, b, c = 5, 0, -3
print(a > 0 and b > 0)          # Ligne 1
print(a > 0 or b > 0)           # Ligne 2
print(not (a > 0))              # Ligne 3
print(a > 0 and not b == 0)     # Ligne 4
print(c < 0 or a + c > 10)      # Ligne 5
print(not (a > 0 and c > 0))    # Ligne 6
```

Exercice 4 *Tarif de bus* Le tarif d'un trajet en bus dépend de l'âge et du statut :

- gratuit pour les moins de 4 ans ;
- tarif réduit (0,80 €) pour les 4–17 ans ou les étudiants ;
- plein tarif (1,60 €) sinon.

Écrire une fonction `tarif_bus(age, etudiant)` où `etudiant` est un booléen.

Solutions des exercices

Corrigé de l'exercice 1.

```
def categorie(age):
    if age < 12:
        return "enfant"
    elif age <= 17:
        return "adolescent"
    else:
        return "adulte"
```

Remarque : pas besoin de tester `age >= 12` dans le `elif` car si on y arrive, c'est que la première condition est fausse (donc `age >= 12`).

Corrigé de l'exercice 2.

```
def type_triangle(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return "invalide"
    elif a == b == c:
        return "équilatéral"
    elif a == b or b == c or a == c:
        return "isocèle"
    else:
        return "quelconque"
```

Principe : on teste d'abord le cas invalide (inégalité triangulaire stricte). Ensuite, l'ordre est important : un triangle équilatéral est aussi isocèle, il faut donc tester l'équilatéral en premier.

Vérification :

- `type_triangle(3, 3, 3)` renvoie `"équilatéral"` ;
- `type_triangle(3, 3, 5)` renvoie `"isocèle"` ;
- `type_triangle(1, 1, 3)` renvoie `"invalide"` car $1 + 1 \leqslant 3$.

Corrigé de l'exercice 3.

1. `False` : `a > 0` est `True` mais `b > 0` est `False`, et `True and False = False`.
2. `True` : `a > 0` est `True`, et `True or False = True`.
3. `False` : `a > 0` est `True`, donc `not True = False`.
4. `False` : `not b == 0` vaut `not True = False`, et `True and False = False`.
5. `True` : `c < 0` est `True`, Python n'évalue même pas la suite (évaluation paresseuse).
6. `True` : `a > 0 and c > 0` vaut `True and False = False`, donc `not False = True`. Autrement dit, la négation de « A et B » revient à « non A ou non B » : il suffit qu'une des deux conditions soit fausse pour que l'ensemble soit nié en `True`.

Corrigé de l'exercice 4.

```python
def tarif_bus(age, etudiant):
    if age < 4:
        return 0
    elif age <= 17 or etudiant:
        return 0.80
    else:
        return 1.60
```

Vérification :

- `tarif_bus(3, False)` renvoie `0`;
- `tarif_bus(15, False)` renvoie `0.80`;
- `tarif_bus(22, True)` renvoie `0.80`;
- `tarif_bus(30, False)` renvoie `1.60`.