Instructions conditionnelles

Bases de Python

Jusqu'ici, nos programmes exécutaient toutes les instructions dans l'ordre, de la première à la dernière. Mais un programme utile doit pouvoir **prendre des décisions** : un distributeur automatique vérifie que l'on a inséré assez d'argent avant de libérer un produit ; un site web affiche un message différent selon que l'on est connecté ou non. C'est le rôle des **instructions conditionnelles**.

1 L'indentation : un concept clé

En Python, l'indentation (les espaces en début de ligne) n'est pas un choix esthétique : elle est **obligatoire** et **significative**. C'est elle qui délimite les **blocs d'instructions**, c'est-à-dire les groupes d'instructions qui vont ensemble.

```
age = 20

if age >= 18:
    print("Vous êtes majeur")      # ← dans le bloc if (4 espaces)
    print("Bienvenue !")           # ← dans le bloc if (4 espaces)
print("Fin du programme")          # ← hors du bloc if (pas d'indentation)
```

Les deux premières instructions `print` ne s'exécutent que si la condition est vraie. La troisième s'exécute toujours, car elle n'est pas indentée dans le bloc.

Règles à retenir :

— on utilise **quatre espaces** par niveau d'indentation (convention standard en Python) ;

— toutes les instructions d'un même bloc doivent avoir **exactement** la même indentation ;

— une indentation incorrecte provoque une erreur `IndentationError`.

```
if True:
    print("ok")
      print("erreur")  # IndentationError : indentation inattendue
```

Dans d'autres langages (C, Java, JavaScript), les blocs sont délimités par des accolades `{ }`. En Python, c'est l'indentation qui joue ce rôle. C'est ce qui rend le code Python particulièrement lisible, à condition de respecter cette règle.

2 L'instruction `if`

L'instruction `if` (« si ») exécute un bloc d'instructions uniquement lorsqu'une condition est **vraie** :

```
temperature = 35

if temperature > 30:
    print("Il fait très chaud !")
    print("Pensez à vous hydrater")
```

```
Il fait très chaud !
Pensez à vous hydrater
```

La condition `temperature > 30` est une **expression booléenne** : elle vaut `True` ou `False`. Ici, `35 > 30` vaut `True`, donc le bloc indenté s'exécute. Les opérateurs de comparaison (`==`, `!=`, `<`, `>`, `<=`, `>=`) et les opérateurs logiques (`and`, `or`, `not`) utilisés dans les conditions sont détaillés dans le cours sur les opérateurs.

Question.
Et si la température vaut `20` : que se passe-t-il ?

Réponse.
Rien ne s'affiche. La condition `20 > 30` vaut `False` : Python saute les deux lignes indentées, sans rien exécuter.

Ajoutons une dernière ligne, sans indentation :

```
temperature = 20

if temperature > 30:
    print("Il fait très chaud !")
    print("Pensez à vous hydrater")
print("Bonne journée")
```

Question.
Quels messages ce programme affiche-t-il ?

Réponse.

```
Bonne journée
```

La condition est fausse : le bloc indenté est sauté. La dernière ligne n'est pas indentée : elle ne fait pas partie du bloc, et s'exécute dans tous les cas.

Attention. ne pas confondre `=` (affectation) et `==` (comparaison d'égalité).

```
x = 5       # affectation : x prend la valeur 5
x == 5      # comparaison : vaut True car x vaut 5
```

③ L'instruction `if...else`

On veut souvent exécuter un bloc si la condition est vraie et un **autre bloc** si elle est fausse. C'est le rôle de `else` (« sinon ») :

```
age = 15

if age >= 18:
    print("Vous êtes majeur")
else:
    print("Vous êtes mineur")
```

```
Vous êtes mineur
```

Ici, la condition `age >= 18` est fausse : Python saute le premier bloc et exécute le bloc placé sous `else`. Exactement l'un des deux blocs s'exécute : jamais les deux, jamais aucun.

④ L'instruction `if...elif...else`

Lorsqu'il y a **plus de deux cas**, on utilise `elif` (contraction de « else if ») pour tester des conditions supplémentaires :

```
note = 14

if note >= 16:
    mention = "Très bien"
elif note >= 14:
    mention = "Bien"
elif note >= 12:
    mention = "Assez bien"
elif note >= 10:
    mention = "Admis"
else:
    mention = "Ajourné"

print(f"Résultat : {mention}")
```

```
Résultat : Bien
```

Python teste les conditions **dans l'ordre**, de haut en bas. Dès qu'une condition est vraie, le bloc correspondant s'exécute et **tout le reste est ignoré**, même les conditions qui seraient vraies elles aussi. Si aucune condition n'est vraie, le bloc `else` s'exécute (s'il est présent). Avec `note = 14` :

Condition	Résultat	Conséquence
`note >= 16`	`False`	on passe à la condition suivante
`note >= 14`	`True`	`mention = "Bien"`, puis on sort de la structure
`note >= 12`	non testée	ignorée
`note >= 10`	non testée	ignorée

L'ordre des conditions compte donc : on commence par la plus exigeante. Voici ce qui arrive si on l'oublie :

```
note = 18

if note >= 10:
    mention = "Admis"
elif note >= 16:
    mention = "Très bien"

print(mention)
```

Question.
Qu'affiche ce programme ?

Réponse.

```
Admis
```

La première condition, `18 >= 10`, est vraie : la mention vaut « Admis », et la condition suivante n'est même pas testée. Un élève ayant 18 obtient donc « Admis » au lieu de « Très bien ».

5 Plusieurs `if` ou `elif` ?

Parfois, plusieurs conditions peuvent être vraies en même temps : un nombre peut être à la fois pair et multiple de 3. On écrit alors plusieurs `if` à la suite. Chaque `if` est testé séparément, quoi qu'il arrive aux autres.

```
n = 12

if n % 2 == 0:
    print("pair")
if n % 3 == 0:
    print("multiple de 3")
```

Question.
Combien de messages s'affichent ?

Réponse.

```
pair
multiple de 3
```

Deux : 12 est pair, et 12 est un multiple de 3. Les deux conditions sont vraies, et les deux `if` sont testés séparément.

Avec `elif`, c'est différent : la première condition est vraie, Python affiche « pair », puis ignore la suite, même si elle est vraie elle aussi.

```python
n = 12

if n % 2 == 0:
    print("pair")
elif n % 3 == 0:
    print("multiple de 3")
```

```
pair
```

À retenir. Plusieurs `if`, quand les conditions sont indépendantes et peuvent être vraies ensemble ; `elif`, quand un seul cas doit être choisi.

6 Combiner des conditions

On combine plusieurs conditions avec les opérateurs logiques. Pour conduire, il faut à la fois avoir au moins 18 ans **et** avoir le permis : `and` exige que les deux conditions soient vraies.

```python
age = 20
permis = True

if age >= 18 and permis:
    print("Vous pouvez conduire")
```

```
Vous pouvez conduire
```

`or` exige qu'au moins une des deux conditions soit vraie, et `not` inverse une condition :

```python
jour = "samedi"

if jour == "samedi" or jour == "dimanche":
    print("C'est le week-end")
if not jour == "lundi":
    print("Ce n'est pas lundi")
```

```
C'est le week-end
Ce n'est pas lundi
```

7 Conditions imbriquées

On peut placer un `if` à l'intérieur d'un autre `if`. Chaque niveau d'imbrication ajoute un niveau d'indentation :

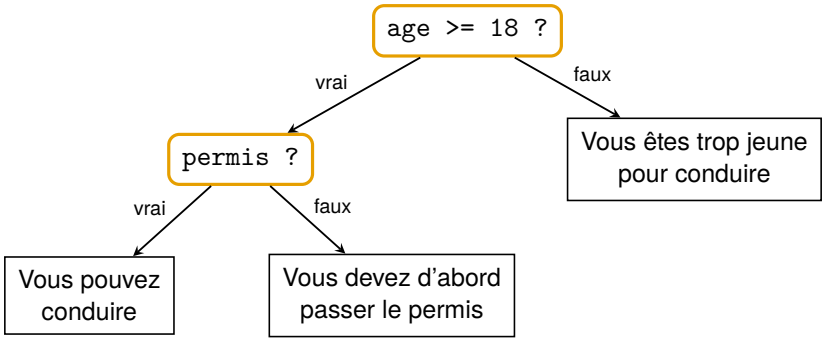
```python
age = 25
permis = True

```

```
if age >= 18:
    if permis:
        print("Vous pouvez conduire")
    else:
        print("Vous devez d'abord passer le permis")
else:
    print("Vous êtes trop jeune pour conduire")
```

```
Vous pouvez conduire
```

Ici, on ne teste si la personne a le permis que si elle est majeure. L'imbrication traduit une **hiérarchie de décisions**, que l'on représente par un **arbre de décision** : chaque question mène, selon la réponse, à une autre question ou à une issue.



On peut souvent simplifier les conditions imbriquées avec `and` :

```
if age >= 18 and permis:
    print("Vous pouvez conduire")
```

Mais la version imbriquée reste préférable lorsque les blocs `else` diffèrent selon le niveau, comme dans l'exemple ci-dessus : chaque issue de l'arbre a son propre message.

⑧ Expressions ternaires

Pour les cas simples, Python permet d'écrire une condition sur **une seule ligne** avec une expression ternaire :

```
age = 20
statut = "majeur" if age >= 18 else "mineur"
print(statut)
```

```
majeur
```

La syntaxe est : `valeur_si_vrai if condition else valeur_si_faux`. C'est équivalent à :

```
if age >= 18:
    statut = "majeur"
```

```
else:
    statut = "mineur"
```

L'expression ternaire est pratique pour des affectations simples, mais il ne faut pas en abuser : dès que la logique devient complexe, un `if...else` classique est plus lisible.

9 Le mot-clé `pass`

Il est parfois utile d'écrire la structure d'un programme avant de remplir les blocs. Mais en Python, un bloc vide provoque une erreur. Le mot-clé `pass` sert de **bloc vide** volontaire :

```
age = 15

if age >= 18:
    pass  # on traitera ce cas plus tard
else:
    print("Vous êtes mineur")
```

```
Vous êtes mineur
```

10 Erreurs fréquentes

Oublier les deux-points. La ligne `if`, `elif` ou `else` doit toujours se terminer par `:`.

```
if age >= 18     # SyntaxError : il manque les deux-points
    print("ok")
```

Confondre `=` et `==`. Écrire `if x = 5` au lieu de `if x == 5` provoque une erreur de syntaxe.

Mauvaise indentation. Mélanger espaces et tabulations, ou ne pas aligner les instructions d'un même bloc, produit une `IndentationError`.

Écrire plusieurs `if` quand un seul cas doit être choisi. Plusieurs `if` successifs sont testés indépendamment : si l'on veut un seul message, il faut `elif`.

```
note = 18

# Incorrect : on voulait un seul message, deux s'affichent
if note >= 10:
    print("Admis")
if note >= 16:
    print("Mention Très bien")
```

```
Admis
Mention Très bien
```

```python
# Correct : un seul message
if note >= 16:
    print("Mention Très bien")
elif note >= 10:
    print("Admis")
```

```
Mention Très bien
```

11 À retenir

1. `if condition:` exécute le bloc indenté seulement si la condition est vraie ; sinon, Python le saute.
2. Le bloc est délimité par l'indentation, de quatre espaces ; une ligne non indentée après le bloc s'exécute toujours.
3. Avec `if` et `else`, exactement un des deux blocs s'exécute.
4. Avec `elif`, Python teste les conditions dans l'ordre et s'arrête à la première vraie : on commence par la plus exigeante.
5. Plusieurs `if` à la suite sont testés séparément : on les choisit quand les conditions sont indépendantes, et `elif` quand un seul cas doit être choisi.
6. `and`, `or` et `not` combinent des conditions.
7. Des conditions imbriquées traduisent une hiérarchie de décisions, qu'un arbre de décision représente.
8. Les erreurs les plus fréquentes : oublier les deux-points, confondre `=` et `==`, mal indenter.

12 Résumé

Instruction	Syntaxe	Rôle
`if`	`if condition:`	Exécute le bloc si la condition est vraie
`else`	`else:`	Exécute le bloc si la condition du `if` est fausse
`elif`	`elif condition:`	Teste une condition supplémentaire
Ternaire	`a if cond else b`	Condition en une ligne
`pass`	`pass`	Bloc vide volontaire

Exercices

Exercice 1 *Parité* **(2 points)** Écrire un programme qui demande un entier à l'utilisateur et affiche s'il est pair ou impair.

Indication : un nombre est pair si le reste de sa division par deux vaut zéro.

Exercice 2 *Catégories d'âge* **(3 points)** Écrire un programme qui demande l'âge de l'utilisateur et affiche sa catégorie :

— moins de 12 ans : « enfant » ;

— de 12 à 17 ans : « adolescent » ;

— de 18 à 64 ans : « adulte » ;

— 65 ans et plus : « senior ».

Exercice 3 *Tarification cinéma* **(3 points)** Un cinéma applique les tarifs suivants :

— moins de 12 ans : 6 € ;

— de 12 à 17 ans : 8 € ;

— de 18 à 64 ans : 12 € ;

— 65 ans et plus : 9 €.

De plus, le mercredi, tous les tarifs sont réduits de 2 €.

Écrire un programme qui demande l'âge et le jour de la semaine, puis affiche le prix du billet.

Exercice 4 *Année bissextile* **(4 points)** Une année est bissextile si elle est divisible par 4, **sauf** si elle est divisible par 100, **sauf** si elle est divisible par 400.

Exemples : 2024 est bissextile (divisible par 4, pas par 100). 1900 n'est pas bissextile (divisible par 100, pas par 400). 2000 est bissextile (divisible par 400).

Écrire un programme qui demande une année et affiche si elle est bissextile ou non.

Exercice 5 *Mention au bac* **(3 points)** Écrire un programme qui demande une note (sur 20) et affiche la mention correspondante :

— note ≥ 16 : « Très bien » ;

— note ≥ 14 : « Bien » ;

— note ≥ 12 : « Assez bien » ;

— note ≥ 10 : « Admis » ;

— sinon : « Ajourné ».

Le programme doit aussi vérifier que la note est comprise entre 0 et 20.

Exercice 6 *Résoudre $ax + b = 0$* **(4 points)** Écrire un programme qui demande les coefficients a et b d'une équation du premier degré $ax + b = 0$ et affiche la solution. Le programme doit traiter tous les cas :

— si $a \neq 0$: l'équation a une unique solution $x = -b/a$;

— si $a = 0$ et $b = 0$: tout réel est solution ;

— si $a = 0$ et $b \neq 0$: l'équation n'a pas de solution.