Chapitre 4 : structure de contrôle, instruction conditionnelle

"Objectifs"

À la fin du chapitre, on doit savoir :

- écrire une condition booléenne simple ou complexe (avec `and`, `ord`, `not`)
- écrire une structure conditionnelle avec un ou plusieurs cas
- indenter correctement un code avec une structure conditionnelle
- corriger un code mal indenté

Écriture de conditions

"Définition 1"

Une **condition** est une expression booléenne, c'est-à-dire une expression dont la valeur est `True` ou `False`.

"Condition simple avec opérateur de comparaison"

Les conditions sont souvent construites avec des opérateurs de comparaison.

Opérateur	Signification
`==`	égal à
`!=`	différent de
`<` (resp. `<=`)	strictement inférieur à (resp. inférieur ou égal à)
`>` (resp. `>=`)	strictement supérieur à (resp. supérieur ou égal à)

"Conditions complexes"

On peut construire des conditions plus complexes avec les opérateurs booléens `and`, `or` et `not`.

Opérateur	Signification	Exemple
`and`	et logique	`(x > 0) and (y > 0)`
`or`	ou logique	`(x > 0) or (y > 0)`
`not`	négation	`not (x > 0)`

"Attention"

En Python, pour tester si une variable `x` est comprise entre `0` et `10` bornes incluses, on peut écrire `0 <= x <= 10` ou encore `(x >= 0) and (x <= 10)` mais pas `x >= 0 and <= 10`.

"Exercice 1"

Dans cet exercice on considère deux variables `x` et `y` de type `int`.

1. Écrire une condition vraie si la variable `x` est égale à 601.

---

2. Écrire une condition vraie si la variable `x` est différente de 601.

---

3. Écrire une condition vraie si la variable `x` n'est pas supérieure ou égale à 700.

---

4. Écrire une condition fausse si la variable `x` est strictement inférieure à 0.

---

5. Écrire une condition vraie si la variable `x` ou la variable `y` est strictement positive.

---

6. Écrire une condition vraie si la variable `x` et la variable `y` sont non nulles.

---

7. Écrire une condition fausse si la variable `x` ou la variable `y` est nulle.

8. Écrire une condition fausse si la variable `x` et la variable `y` sont nulles.

Instruction(s) conditionnelle(s)

"Définition 2"

Pour contrôler le flux d'exécution d'un programme, on a parfois besoin d'exécuter un bloc d'instructions uniquement si une condition est vérifiée.

L'instruction conditionnelle `if` permet de réaliser ce contrôle avec la syntaxe suivante :

```
bloc_avant
if condition1:
    bloc_si_condition1_vraie
bloc_apres
```

Toutes les instructions du bloc contrôlé par l'instruction conditionnelle `if condition1:` sont décalés du même nombre d'espaces (4 par convention ou un tabulation) par rapport au `if` et aux blocs qui s'exécutent avant ou après l'instruction conditionnelle. Ce décalage s'appelle une indentation. Chaque niveau d'indentation supplémentaire correspond à un embranchement logique du programme.

Pour annoncer que les instructions suivantes vont être indentées, la ligne de l'instruction conditionnel `if condition1:` se termine par le caractère `:`.

"Exemple 1"

Le programme suivant demande l'âge de l'utilisateur.

- Si la condition `age >= 18` est vraie, alors le message `"Majeur"` est affiché.
- Sinon, le message `"Mineur"` est affiché.

Dans tous les cas, l'instruction `print("Fin du programme")` est exécutée, car elle n'est pas indentée dans le bloc du `if` ou du `else`.

```
age = int(input("Votre âge ?"))
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
print("Fin du programme")
```

"Définition 3"

On a vu que l'instruction conditionnelle `if condition1:` exécute `bloc_si_condition1_vraie` si `condition1` est vraie et ne fait rien sinon.

Parfois, la vérification de `condition1` peut nous conduire à exécuter `bloc_si_condition1_fausse` si `condition1` est fausse. On ajoute alors à l'instruction conditionnelle `if condition1:` une instruction conditionnelle `else:` et la syntaxe devient :

```
bloc_avant
if condition1:
    bloc_si_condition1_vraie
else:
    bloc_si_condition1_fausse
bloc_apres
```

On peut aussi avoir plusieurs conditions à vérifier, qui s'excluent mutuellement, la vérification de chaque condition nous conduisant à exécuter un bloc. On peut ajouter alors à l'instruction conditionnelle `if condition1:` des instructions conditionnelles `elif conditionX:` qu'on peut terminer par une instruction `else:` pour l'exécution d'un bloc si toutes les conditions précédentes n'ont pas été vérifiées.

```
bloc_avant
if condition1:
    bloc_si_condition1_vraie
elif condition2:#si (not condition1) and condition2
    bloc_si_condition2_vraie
elif condition3:#si (not condition1) and (not condition2) and condition3
    bloc_si_condition3_vraie
else:#si (not condition1) and (not condition2) and (not condition3)
    bloc_restant
bloc_apres
```

"Attention"

Dans une structure `if ... elif ... else`, la vérification des conditions s'effectue en cascade et un seul bloc est exécuté : le premier dont la condition est vraie. L'ordre des conditions est donc important.

"Exemple 2"

Le programme suivant affiche un tarif en fonction de l'âge.

```
age = int(input("Votre âge ? "))

if age < 6:
    tarif = 0
elif age < 18:
    tarif = 5
else:
    tarif = 10

print("Tarif :", tarif, "euros")
```

Si `age` vaut `4`, la variable `tarif` vaut `0`.
Si `age` vaut `15`, la variable `tarif` vaut `5`.
Si `age` vaut `25`, la variable `tarif` vaut `10`.

"Exercice 2"

On considère le programme suivant.

```
age = int(input("Votre âge ? "))

if age < 12:
    categorie = "enfant"
elif age < 18:
    categorie = "adolescent"
else:
    categorie = "adulte"

print(categorie)
```

1. Quelle valeur est affichée si l'utilisateur saisit `10` ?

---

2. Quelle valeur est affichée si l'utilisateur saisit `15` ?

---

3. Quelle valeur est affichée si l'utilisateur saisit `20` ?

---

4. Pourquoi l'ordre des conditions est-il important ?

"Exercice 3"

Voici la spécification des résultats du bac :

- si la moyenne du candidat est strictement inférieure à `8`, alors il est *refusé* ;
- sinon, si sa moyenne est strictement inférieure à `10`, alors il passe *l'oral du second groupe* ;
- sinon, si sa moyenne est strictement inférieure à `12`, alors il est admis avec mention *passable* ;
- sinon, si sa moyenne est strictement inférieure à `14`, alors il est admis avec mention *assez bien* ;
- sinon, si sa moyenne est strictement inférieure à `16`, alors il est admis avec mention *bien* ;
- sinon, il est admis avec mention *très bien*.

Un élève a écrit un programme pour afficher le résultat du bac à partir de la moyenne :

```
moyenne = float(input("Moyenne ? "))
if moyenne < 8:
    res = "refusé"
elif moyenne < 10:
res = "second groupe"
else:
    res = "mention très bien"
print(res)
```

1. À l'exécution, il obtient le message d'erreur suivant. Proposer une correction.

```
res = "second groupe"
^
IndentationError: expected an indented block
```

---

2. L'erreur de syntaxe précédente a été corrigée et le programme s'exécute sans erreur.
   Quels sont les types respectifs :
   - de la variable `moyenne` ?
   - de l'expression `moyenne < 10` ?
   - de la variable `res` ?

   ---

3. Quelle est la valeur affichée en sortie par le programme pour une moyenne de `13` ?

   ---

4. Si cette valeur est incorrecte, proposer une correction du programme pour qu'il respecte la spécification des résultats du bac.

Erreurs

> **"Erreur de type `IndentationError`"**
>
> L'erreur `IndentationError` apparaît lorsque l'indentation du programme ne respecte pas la syntaxe Python.
> Elle se produit souvent lorsqu'un bloc d'instructions attendu après une ligne terminée par `:` n'est pas correctement indenté.
>
> Exemple de programme mal indenté :

```
age = int(input("Votre âge ?"))
if age > 18:
print("Majeur") # instruction mal indentée
else:
    print("Mineur")
```

"À retenir sur l'indentation"

En Python, l'indentation n'est pas seulement une question de présentation. Elle fait partie de la syntaxe du langage. Deux instructions qui appartiennent au même bloc doivent avoir exactement le même niveau d'indentation.

Dans d'autres langages, comme `C` ou `Javacript`, l'indentation sert juste à la présentation et les blocs sont délimités par des accolades : `{bloc}`.

"Exercice 4"

Pour chaque programme, dire s'il est correctement indenté.

Si le programme est mal indenté, proposer une correction.

Programme A

```
age = int(input("Âge ? "))
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
print("Fin")
```

Programme B

```
age = int(input("Âge ? "))
if age >= 18:
print("Majeur")
else:
    print("Mineur")
print("Fin")
```

Programme C

```
note = float(input("Note ? "))
if note >= 10:
    print("Admis")
    print("Bravo")
else:
    print("Refusé")
```

Programme D

```
note = float(input("Note ? "))
if note >= 10:
    print("Admis")
print("Bravo")
else:
    print("Refusé")
```

Programme E

```
temperature = float(input("Température ? "))
if temperature < 0:
    etat = "solide"
elif temperature < 100:
    etat = "liquide"
else:
    etat = "gazeux"
print(etat)
```
