Les structures conditionnelles en Python

Table des matières

Instructions conditionnelles : généralités

Les instructions conditionnelles permettent de contrôler l'exécution de différentes parties d'un programme selon la valeur d'une expression **booléenne** (qui peut être `True` ou `False`).

1 - L'instruction `if`

La syntaxe de base de l'instruction `if` est :

```
if expression:
    bloc_instructions_1
bloc_instructions_2
```

- **expression** est évaluée en tant que `True` ou `False`.
- Si **expression** est `True`, le bloc **bloc_instructions_1** est exécuté, sinon il est ignoré.
- Dans tous les cas, le **bloc_instructions_2** sera exécuté après, qu'il y ait eu exécution de **bloc_instructions_1** ou non.

Exemple : l'expression du `if` est `True`

```
nombre = 150
if nombre % 3 == 0:
    print(f"{nombre} est divisible par 3")
print("Fin des vérifications")
```

Résultat :

```
150 est divisible par 3
Fin des vérifications
```

Exemple : l'expression du `if` est `False`

```
nombre = 151
if nombre % 3 == 0:
    print(f"{nombre} est divisible par 3")
print("Fin des vérifications")
```

Résultat :

```
Fin des vérifications
```

2 - L'instruction `if ... else`

L'instruction `if ... else` permet d'ajouter une alternative dans le cas où l'expression du `if` est `False`. Sa syntaxe est la suivante :

```
if expression:
    bloc_instructions_1
else:
    bloc_instructions_2
bloc_instructions_3
```

- Si **expression** est `True`, **bloc_instructions_1** est exécuté, sinon **bloc_instructions_2** est exécuté.
- Dans tous les cas, **bloc_instructions_3** sera exécuté après.

Exemple :

```
nombre = 7
if nombre % 2 == 0:
    print(f"{nombre} est pair")
else:
    print(f"{nombre} est impair")
```

Résultat :

```
7 est impair
```

3 - L'instruction `if ... elif ... else`

Lorsque vous avez besoin de tester plusieurs alternatives, vous pouvez utiliser l'instruction `if ... elif ... else`. **`elif`** signifie "else if" et permet d'ajouter des conditions supplémentaires après un `if`.

La syntaxe est :

```
if expression1:
    bloc_instructions_1
elif expression2:
    bloc_instructions_2
else:
    bloc_instructions_3
bloc_instructions_4
```

- Si **expression1** est `True`, **bloc_instructions_1** est exécuté.
- Si **expression1** est `False` et **expression2** est `True`, **bloc_instructions_2** est exécuté.
- Si toutes les expressions sont `False`, **bloc_instructions_3** est exécuté.
- Dans tous les cas, **bloc_instructions_4** sera exécuté après, qu'importe le chemin pris dans la condition.

Exemple :

```
nombre = 44
if nombre % 5 == 0:
    print(f"{nombre} est un multiple de 5")
elif nombre % 4 == 0:
    print(f"{nombre} est un multiple de 4")
else:
    print(f"{nombre} n'est ni un multiple de 5 ni de 4")
```

Résultat :

```
44 est un multiple de 4
```

4 - Explications supplémentaires

- L'exécution des blocs d'instructions dépend des valeurs **booléennes** des expressions. Vous pouvez tester autant de conditions que nécessaire avec des **`elif`**.
- Il est important de bien structurer votre code avec une **indentation correcte** (4 espaces) pour éviter les erreurs d'exécution.

5 - L'instruction `match`

L'instruction `match` est une nouvelle fonctionnalité introduite dans **Python 3.10**. Elle permet d'effectuer des comparaisons entre une variable et plusieurs cas possibles, un peu comme un "`switch-case`" dans d'autres langages de programmation. C'est une manière élégante de gérer plusieurs conditions de manière lisible et claire.

La syntaxe de base est la suivante :

```
match variable:
    case valeur1:
        bloc_instructions_1
    case valeur2:
        bloc_instructions_2
    case _:
        bloc_instructions_par_defaut
```

- **variable** : la valeur que l'on souhaite comparer aux différents cas.
- **case** : chaque condition ou "cas" est testé pour voir s'il correspond à la variable.
- _ : le cas par défaut, qui est exécuté si aucun des cas précédents ne correspond.

Exemple : Cas de réponse HTTP

Imaginons que vous souhaitiez traiter différents codes de réponse HTTP :

```
response_code = 400
match response_code:
    case 200:
        print("Tout va bien !")
    case 301:
        print("Redirection vers une autre page.")
    case 400:
        print("Erreur : Mauvaise requête.")
    case 500:
        print("Erreur serveur.")
    case _:
        print("Code de réponse inconnu.")
```

Résultat :

```
Erreur : Mauvaise requête.
```

Explications

- Si la valeur de la variable correspond à l'un des cas définis (par exemple, ici, `400`), le bloc d'instructions correspondant à ce cas est exécuté.
- Le caractère spécial _ est utilisé pour gérer le "cas par défaut" ou "autre cas", si aucun des autres cas ne correspond.

Cela permet de rendre plus lisible le traitement d'une variable qui peut prendre plusieurs valeurs distinctes, sans devoir écrire une série d'instructions `if ... elif ... else`.

Autre exemple avec variables

```
animal = "chat"
match animal:
    case "chien":
        print("C'est un chien.")
    case "chat":
        print("C'est un chat.")
    case "oiseau":
        print("C'est un oiseau.")
    case _:
        print("Animal inconnu.")
```

Résultat :

```
C'est un chat.
```

L'instruction `match` est particulièrement utile lorsque vous avez plusieurs cas spécifiques à traiter et que vous souhaitez éviter des chaînes complexes d'instructions `if...elif...else`.