IngeSUP - Cours 02 - Les structures conditionnelles

Sommaire

Objectifs

- Comprendre les opérateurs de comparaison ;
- Introduire les opérateurs booléens ;
- Apprendre à implémenter une instruction conditionnelle.

Vidéos

Le sujet abordé dans ce notebook de cours est traité sur ionisX dans la capsule vidéo suivante :

- Module 5 : Structures de contrôle en Python (https://courses.ionisx.com/courses/ref/m122/x/courseware/d1d662e54cf64d7d91aea5d1a45e1904/)

Introduction

Les programmes que nous avons écrits jusqu'ici s'exécutent de manière très ordonnée. Ce n'est pas ainsi que fonctionnent les programmes dans le monde réel.

Parfois, nous souhaitons exécuter un ensemble d'instructions **uniquement lorsque certaines conditions sont remplies**.

Pour gérer ce type de situation, les langages de programmation fournissent des **instructions spéciales appelées instructions de contrôle**.

Avant de voir les structures conditionnelles utilisées via l'instruction `if` , il est important de d'introduire les **opérateurs de comparaison** et les **opérateurs booléens**.

Les opérateurs de comparaison

Dans un programme, nous avons régulièrement besoin de comparer deux variables, voir si par exemple l'une est plus grande que l'autre ou égale.

Pour cela, nous utilisons des opérateurs de comparaison : `<` , `<=` , `>` , `>=` and `==` .

Testons ci-dessous si une variable `a` est plus grande ou plus petite qu'une variable `b` :

Entrée [ ]:

```
a = 12.0
b = 9.7
print(a < b)
print(a > b)
```

Les égalités sont testées à partir des opérateurs ' == ' (égal) et `!=` (différent de). Ces opérateurs sont utilisés pour tester l'égalité de deux valeurs. Ci-dessous quelques exemples :

Entrée [ ]:

```
a = 12
b = -9
c = 14

# Teste si a est égale à b
print("Est-ce-que a est égale à b?")
print(a == b)
```

Entrée [ ]:

```
# Teste si a est égale à c
print("Est-ce-que a est égale à c?")
print(a == c)

# Teste si a est différent de c
print("Est-ce-que a est différent de c?")
print(a != c)
```

Entrée [ ]:

```
# Teste si a est inférieure ou égale à b
print("Est-ce-que a est inférieure ou égale à b?")
print(a <= b)

# Teste si a est inférieure ou égale à c
print("Est-ce-que a est inférieure ou égale à c?")
print(a <= c)
```

Entrée [ ]:

```
# Teste si deux couleurs sont les mêmes
couleur0 = 'bleu'
couleur1 = 'vert'
print("Est-ce-que la couleur0 est la même que couleur1?")
print(couleur0 == couleur1)
```

Les opérateurs booléens

Jusqu'à maintenant, nous n'avons utilisé que des comparaisons uniques à l'aide des opérateurs de comparaisons. Les opérateurs booléens permettent les comparaisons multiples à partir des opérateurs `and`, `or` et `not`.

Les opérateurs `and` et `or` considèrent un booléen de chaque côté de l'expression et le code

```
X and Y
```

évaluera `True` si `X` **et** `Y` sont tous les deux vrai ou évaluera `False` dans le cas contraire.

Le code

```
X or Y
```

évaluera `True` si `X` **ou** `Y` est vrai ou évaluera `False` dans le cas contraire.

Voici deux exemples :

Entrée [ ]:

```
# Teste si 10 < 9 (false) et 15 < 20 (true) -> false
print(10 < 9 and 15 < 20)
```

Entrée [ ]:

```
# Teste si 10 < 9 (false) ou 15 < 20 (true) -> true
print(10 < 9 or 15 < 20)
```

Il est important de noter que les opérateurs de comparaisons (`>=`, `<=`, `<` et `>`) sont évalués avant les opérateurs booléens (`and`, `or`)

L'opérateur `not` effectue une négation logique sur l'expression testée :

Entrée [ ]:

```
# Est-ce-que 12 *not* (n'est pas) plus petit que 7 -> true
a = 12
b = 7
print(not a < b)
```

N'utilisez seulement ' `not` ' que si l'expression est facile à lire (i.e. à comprendre) !

Par exemple :

Entrée [ ]:

```
print(not 12 == 7)
```

N'est pas forcément clair. Le mieux est de tester :

Entrée [ ]:

```
print(12 != 7)
```

Dans la plupart des cas, nous ne traiterons au maximum que deux comparaisons. Dans certains cas, vous serez amenés à évaluer plus de deux comparaisons.

Vous pourrez contrôler l'ordre des comparaisons en utilisant des parenthèses.

Par exemple, si nous voulons tester si un nombre est strictement compris entre **100** et **200** ou entre **10** et **50** :

Entrée [ ]:

```
value = 150.5
print ((value > 100 and value < 200) or (value > 10 and value < 50))
```

Les deux parenthèses sont évaluées en premier (chacune évalue `True` ou `False`), et ensuite le `or` vérifie si l'une des deux est vraie.

Maintenant que nous sommes en mesure de comparer des variables, nous pouvons explorer les instructions conditionnelles.

L'instruction `if`

L'instruction est exécutée si une condition est remplie.

```
if condition:
    < instruction indentée 1>
    < instruction indentée 2>

< instruction non indentée>
```

La première ligne de l'instruction **if** est une expression booléenne évaluée à True ou False. Dans la ligne suivante, nous avons un bloc d'instructions. Un bloc est simplement un ensemble d'une ou plusieurs instructions.

Lorsqu'un bloc d'instructions est à l'intérieur de l'instruction `if` , Il est connu comme **bloc if**.

Notez que chaque instruction à l'intérieur du bloc if doit être **indentée** du même nombre d'espaces.

Entrée [ ]:

```
x = int(input("Entrez le nombre 2, soyez cools."))
if x == 2:
    print("Vous avez joué le jeu !")
```

Comment ça marche ?

Lorsque l'instruction if est exécutée, la condition est testée. Si la condition est vraie, toutes les instructions du bloc `if` sont exécutées. Par contre, si la condition est fausse, toutes les instructions du bloc `if` sont ignorées.

Les instructions suivies par la clause `if` qui ne sont pas indentées **n'appartiennent pas au bloc `if`** .

Par exemple, dans l'exemple précédent **< instruction non indentée>** ne fait pas partie du bloc `if` .

En conséquence, il sera toujours exécuté, que la condition de la clause *if* soit vraie ou fausse.

Entrée [ ]:

```
a=int(input("Saisir un nombre : "))
if a>0:
    print("le nombre est positif")
print("bye")    # Instruction non indentée donc toujours exécutée
```

> ⚠ **Attention**
>
> N'oubliez pas de mettre les **:** juste après le condition du *if*
>
> N'oubliez pas d'utiliser l'indentation pour écrire le bloc d'instruction qui appartient au *if* !

Entrée [ ]:

```
a=int(input("Saisir un nombre : "))
if a>0:
    print("le nombre est positif : ")
    c=a+5
    print(c)
print("bye")    # Instruction non indentée
```

> 📝 **Note**
>
> Les programmes que nous avons écrits jusqu'à présent se terminent sans montrer aucune réponse à l'utilisateur si la condition est fausse.

> La plupart du temps, nous voulons montrer à l'utilisateur une réponse même si la condition est fausse. Nous pouvons facilement le faire en utilisant la déclaration ***if-else***

L'instruction `if-else`

Une instruction **if-else** exécute un ensemble d'instructions lorsque la condition est vraie et un ensemble d'instructions différent lorsque la condition est fausse. De cette manière, une instruction **if-else** nous permet de suivre **deux lignes de conduite**.

```
if condition:
    < instruction indentée 1>
    < instruction indentée 2>
else:
    < instruction indentée 3>
    < instruction indentée 4>

< instruction non indentée>
```

Entrée [ ]:

```
x = 2
if x == 2:
    print("Le test est vrai !")
else:
    print("Le test est faux !")
```

Comment ça marche ?

Lorsque l'instruction **if-else** est exécutée, la condition est testée et si elle est évaluée à **True**, les instructions à l'intérieur du bloc if sont exécutées. Toutefois, si la condition est **False**, les instructions du bloc else sont exécutées.

Entrée [ ]:

```
a=int(input("Saisir un nombre : "))
if a>=0:
    print("le nombre est supérieur ou égal à 0")
else:
    print("le nombre est négatif")

print("bye")
```

> ⚠ **Attention**
>
> Dans une instruction **if-else**, assurez-vous toujours que les clauses if et else sont correctement alignées. Sinon, une erreur de syntaxe sera générée.

L'instruction `if-elif-else`

L'instruction **if-else** exécute deux codes différents selon que la condition est **True** ou **False**.

Parfois, un choix doit être fait **parmi plus de deux possibilités**.

if - elif - else vous permet de vérifier plusieurs conditions et d'exécuter différentes instructions.

```
if condition_1:
    < instruction indentée 1>
    < instruction indentée 2>
elif condition_2:
    < instruction indentée 3>
    < instruction indentée 4>
elif condition_3:
    < instruction indentée 5>
    < instruction indentée 6>
...
else:
    < instruction indentée 7>
    < instruction indentée 8>

< instruction indentée 1>
```

Entrée [ ]:

```
x = int(input("Entrez une valeur pour x: "))

if x > 0.0:
    print('La valeur initiale de x est supérieure à 0')
    x -= 20.0    # equivalent de x = x - 20.0
elif x < 0.0:
    print('La valeur initiale de x est inférieure à 0')
    x += 21.0    # equivalent de x = x + 21.0
else:
    print('La valeur initiale de x n\'est ni supérieure à 0, ni inférieure à 0, cela doit é

# Affiche la nouvelle valeur de x
print("La nouvelle valeur de x est:", x)
```

Regardons maintenant plus en détail cette instruction conditionnelle. Celle-ci commence par un `if` , suivie de l'expression à tester, suivie par ' : '

```
if x > 0.0:
```

Ensuite, du code est **indenté**, code qui sera exécuté uniquement si la condition `x > 0.0` est vérifié (est vraie). Une fois le code indenté exécuté, le programme quitte la structure conditionnelle.

```
        print('La valeur initiale de x est supérieure à 0)
        x -= 20.0
```

Si la condition testée est fausse, le programme vient tester la condition du `elif` (else if).

```
    elif x < 0.0:
        print('La valeur initiale de x est inférieure à 0')
        x += 21.0
```

Si la condition `elif` est vérifiée, le code indenté correspondant est exécuté et le programme quitte la structure conditionnelle.

Si aucune des conditions précédentes n'est vérifiée, le code indenté après l'instruction `else` est exécuté puis le programme quitte la structure conditionnelle.

```
    else:
        print('La valeur initiale de x n\'est ni supérieure à 0, ni inférieure à 0, ce
    la doit être 0 !')
```

> 📝 **Note**
>
> Vous pouvez mettre autant d'instruction **elif** que vous voulez et le **else** final n'est pas obligatoire.

Exercices de TD

Vous pouvez maintenant vous exercer à partir du notebook [TD 02 - Les structures conditionnelles (../TD/TD%2002%20-%20Les%20structures%20conditionnelles.ipynb)](../TD/TD%2002%20-%20Les%20structures%20conditionnelles.ipynb).