I4 - Utiliser une instruction conditionnelle

Les booléens

Définition 1

Le type booléen se nomme `bool` en Python. Il permet de représenter les valeurs de vérité : `True` (vrai) et `False` (faux).

On peut créer des expressions booléennes à l'aide des symboles :

Symbole Python	Signification	Mathématique
`==`	est égal à	$=$
`!=`	est différent de	$\neq$
`<=`	est inférieur ou égal à	$\leqslant$
`<`	est strictement inférieur à	$<$
`>=`	est supérieur ou égal à	$\geqslant$
`>`	est strictement supérieur à	$>$
`in`	appartient à	$\in$
`not in`	n'appartient pas à	$\notin$

et des connecteurs logiques :

Symbole Python	Signification	Mathématique
`and`	et	ET
`or`	ou	OU
`not`	négation	NON

Remarque : Le symbole pour tester une égalité est `==` et non `=`. En effet, le symbole `=` sert à l'affectation d'un variable. Par exemple :

```
>>> x = 1    # Création de la variable x et affectation
>>> x == 2   # Test d'égalité, x vaut toujours 1
False
>>> x = 2    # Nouvelle affectation, x vaut maintenant 2
>>> x == 2   # Test d'égalité, x vaut toujours 2
True
```

On voit bien ici que l'on a besoin de deux symboles différents car on veut exprimer deux instructions différentes.

Instructions conditionnelles

Il arrive souvent, au cours d'un programme, de vouloir exécuter certaines instructions mais seulement si une ou plusieurs conditions sont réunies, ou bien de vouloir exécuter des instructions différentes selon la valeur d'une variable.

Pour cela, nous aurons besoin de l'instruction conditionnelle.

Il y a plusieurs types d'instructions conditionnelles :

Test simple

```
if condition:
    instructions    # exécutées si et seulement si condition est vraie
```

Test simple avec alternative

```
if condition:
    instructions    # exécutées si et seulement si condition est vraie
else:
    instructions    # exécutées si et seulement si condition est fausse
```

Test multiple

```
if condition1:
    instructions    # exécutées si et seulement si condition1 est vraie
elif condition2:
    instructions    # exécutées si et seulement si condition1 est fausse
                    # et condition2 est vraie
```

Test multiple avec alternative

```
if condition1:
    instructions    # exécutées si et seulement si condition1 est vraie
elif condition2:
    instructions    # exécutées si et seulement si condition1 est fausse
                    # et condition2 est vraie
else:
    instructions    # exécutées si et seulement si condition1 et condition2 sont fausses
```

où

- ⋆ `condition`, `condition1`, `condition2` sont des expressions booléennes
- ⋆ `instructions` sont des suites d'instructions **indentées** (voir partie Indentiation plus bas) qui ne seront exécutées que sous certaines conditions.

Remarque :

- ⋆ `else` se traduit par « sinon ».
- ⋆ `elif` se traduit par « sinon si ».
- ⋆ `else` n'est pas obligatoire : un principe général à garder en tête est que si on n'a rien à demander à Python, alors il ne faut rien lui dire !

Indentation

Il est très important d'écrire les instructions du test décalées d'une tabulation (4 espaces) par rapport à la ligne contenant le `if`. Ce décalage s'appelle l'indentation. Il sert à délimiter les instructions qui ne doivent être exécutés que sous la condition de celles qui doivent être toujours exécutées.

Par exemple

```
if x > 0:
    print("Je ne m'affiche que si x > 0.")
print("Je m'affiche quoi qu'il arrive.")
```

Exemples

⋆ **Expressions booléennes**

```
>>> x = 12
>>> y = 5
>>> x > y
True
>>> x == y
False
>>> x % y == 0
False
>>> x > x**2 and y < x
True
```

⋆ **Instructions conditionnelles**

```
delta = b**2-4*a*c

if delta == 0:
    print("C'est une identité remarquable")

if delta > 0:
    print("Il y a deux racines")
else:
    print("Il n'y a pas deux racines")

if delta > 0:
    print("Il y a deux racines")
elif delta == 0:
    print("Il y a une racine")
else:
    print("Il n'y a pas de racine")
```

Erreurs fréquentes

▶ **Écrire `true` au lieu de `True` et `false` au lieu de `False`**

Python pensera qu'on utilise une variable appelée `true` ou `false` qui n'existe probablement pas et produira une erreur de nom `NameError`.

▶ **Inverser les symboles `=!` au lieu de `!=`, `=>` au lieu de `>=`, etc...**

Cela produira une erreur de syntaxe `SyntaxError`. On peut retenir que tout symbole combiné avec le `=` est **toujours avant celui-ci** : `!=`, `+=`, `-=`, `*=`, `<=`, `>=`, `==`, ...

▶ **Confusion entre `=` et `==`**

```
if x = 1:
    print(x)
```

donnera une erreur de syntaxe `SyntaxError`.

▶ **Mettre une condition après un `else`**

```
if x == 1:
    print(x)
else x != 1:
    print(x + 1)
```

donnera une erreur de syntaxe `SyntaxError`. En effet, mettre une condition après `else` est incohérent, `else` signifiant sinon et non sinon si.