PÔLE CHATEAUBRIAND JOLIOT-CURIE

Fiche 04 - Structures conditionnelles, booléens et expressions logiques

Informatique - Syntaxes - e618-710888

Objectifs : connaître les types booléens (`bool`), les opérateurs relationnels et logiques et la syntaxe d'une structure conditionnelle.

A Booléens, opérateurs logiques et expressions logiques

Définition : les **booléens**, type `bool`, sont des objets ne pouvant prendre que deux valeurs : `True` ou `False`, pour `VRAI` ou `FAUX`.

Ces valeurs, résultats d'expressions, sont utilisées dans les structures conditionnelles `if` et les boucles conditionnelles `while`.

A.1 Opérateurs relationnels : opérateurs de **comparaison**, ils retournent des booléens. Ces opérateurs sont :

- `==` et `!=` pour tester si deux objets ont la même valeur ou s'ils ont des valeurs différentes ;
- `>`, `<`, `<=`, `>=` pour tester si le terme de gauche est plus grand, plus petit, plus petit ou égale, ou, pour le dernier, plus grand ou égale que celui de droite.

Si la condition est valide, l'opérateur retourne `True`, `False` sinon. Les objets doivent évidemment être comparables. C'est le cas des entiers et flottants (mais aussi des chaînes de caractères).

Exemples :

```
>>> 18 > 9
True
>>> 18 == 19.4
False
>>> 1. == 1
True
>>> 2 != 4/2
False
```

On remarque que l'on peut comparer des flottants avec des entiers.

Priorité : les opérateurs de comparaison ont une priorité plus faible que `+`, `-`, `*`, `/`, `//` ou `%`. Ils sont évalués après les opérations arithmétiques usuelles.

Propriété : les opérateurs peuvent être **enchaînés** pour générer des expressions du type `X < Y <= Z`.

```
>>> 1 < 2 < 3
True
>>> 1 < 1 < 3
False
```

A.2 Opérateurs logiques : opérateurs s'appliquant sur des booléens. Il s'agit de `and`, `or` et `not`. Le résultat est aussi un booléen.

L'opérateur `not` est unaire, et correspond au **non logique** : `not True` retourne `False` et `not False` retourne `True`.

Propriété : les opérateurs `and` et `or` sont dits **paresseux** ; le deuxième terme n'est évalué que si cela est nécessaire.

— `a and b` : **le résultat est vrai** si et seulement **si les deux valeurs sont vraies**. Conséquence : si `a` est faux, quelque soit la valeur booléenne de `b` le résultat de l'expression est `False`. Dans ce cas, l'expression `b` n'a pas besoin d'être évaluée et ne l'est pas ;

— `a or b` : **le résultat est vrai** si et seulement **si au moins une des valeurs est vraie**. Conséquence : si `a` est vraie, quelque soit la valeur booléenne de `b` le résultat de l'expression est `True`. L'expression `b` n'est donc pas évaluée.

L'expression non évaluée peut être une expression qui aurait donnée une erreur : `True or 1/0` retourne `True` alors que la division de 1 par 0 retourne une erreur.

Priorité : les opérateurs logiques ont une priorité très faible. Ils sont évalués en dernier dans l'ordre : `not`, puis `and` puis `or`.

B Bloc d'instructions

Comme vu précédemment, les instructions d'un programme s'exécutent les unes à la suite des autres. L'exécution commence par la première instruction du script (ou de la cellule), puis passe à la seconde, et ainsi de suite.

Cependant, il faut souvent modifier l'ordre d'exécution des instructions. Il s'agit de sauter un passage du programme, de répéter plusieurs fois un groupe d'instructions, de répéter un autre groupe tant qu'une condition n'est pas vérifiée, ou encore d'arrêter le programme avant la fin du script.

En Python, les instructions qui font partie d'un même groupe doivent impérativement avoir la même indentation, c'est à dire le même décalage par rapport au bord gauche du script. On définit ainsi un bloc d'instructions : toutes les lignes du programme qui se suivent et qui présentent la même indentation font partie d'un même bloc d'instructions.

Principe :

— un nouveau bloc d'instruction débute après une **ligne d'entête** qui se termine par deux-points `:` ;

— les deux-points marquent l'ouverture d'un nouveau bloc d'instructions ;

— le nouveau bloc d'instruction est décalé vers la droite par rapport au précédent (de 4 espaces en général) ;

— au sein d'un bloc, le niveau d'indentation reste le même ;

— les blocs sont imbriqués : à la fin d'un bloc, on retombe dans le bloc ouvert précédemment.

Remarque : la boucle `for` et la définition d'une fonction à l'aide de l'instruction `def` respectent ces principes.

```
1 n=10 # bloc 1
2 def somme(n): # ligne d'entête se finissant par deux-points et ouvrant un bloc 2
3     s, p = 0
4     for i in range(1, n+1): # ligne d'entête se finissant par : et ouvrant un bloc 3
5         s = s + i
6         p = p * i # dernière ligne du bloc 3
7     res = p / s # suite du bloc 2
8     return res # dernière ligne du bloc 2
9 print(somme(n)) # suite du bloc 1
```

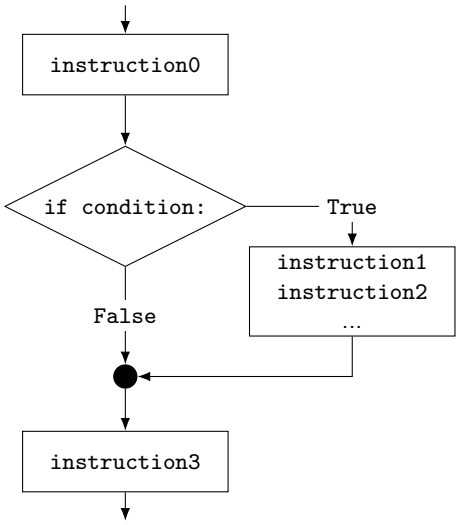
Sur le même principe, il nous reste à revoir les structures conditionnelles et les boucles conditionnelles (boucle TANT QUE, `while`).

C Structures conditionnelles

Les **structures conditionnelles** permettent d'exécuter un bloc d'instructions uniquement si une **condition est vérifiée**. Même si python est un peu plus souple, on s'assurera que la **condition** est une **expression logique** donnant un **booléen** comme résultat.

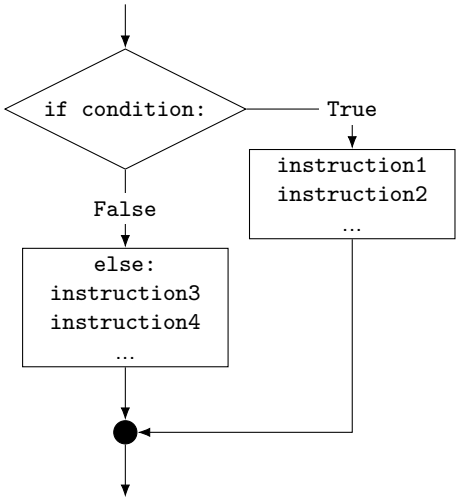
C.1 Syntaxe des structures conditionnelles : plusieurs syntaxes sont possibles, mais débutent toutes avec le mot clef `if`. La structure peut aussi utiliser un ou plusieurs mot clef `elif` et un unique mot clef `else`.

La syntaxe la plus simple utilise seulement le mot clef `if`. Ci-dessous, après l'instruction 0, si la condition vaut vrai, les instructions 1 et 2 sont exécutées, puis l'instruction 3. Par contre, si la condition vaut faux, l'instruction 3 est exécutée directement après l'instruction 0.




```
1 instruction0
2 if condition: # entête avec condition logique
3     instruction1 # bloc si la condition vaut vrai
4     instruction2
5 instruction3
```

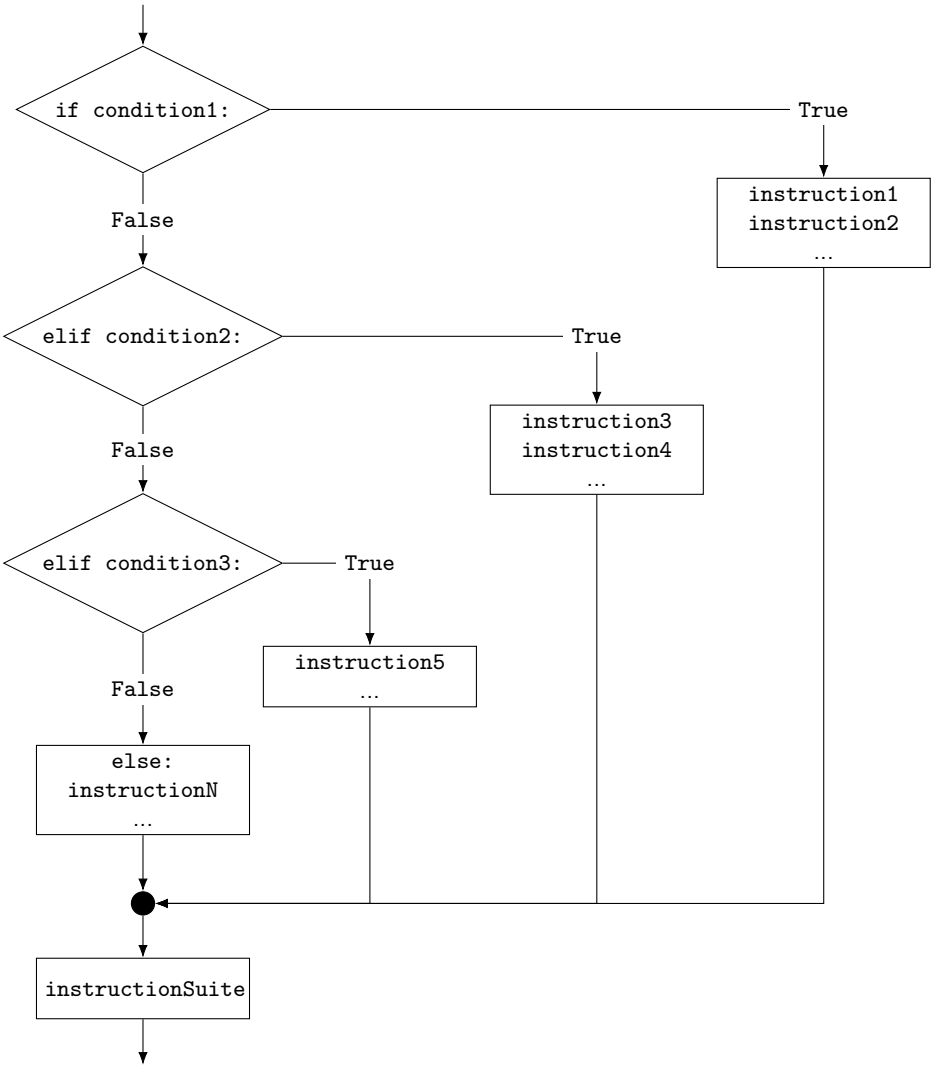
Le mot clef `else` permet de définir un bloc d'instruction qui ne sera exécuté que si la condition vaut faux :




```
1 if condition: # entête avec condition logique
2     instruction1 # bloc si la condition vaut vrai
3     instruction2
4 else:
5     instruction3 # bloc si la condition vaut faux
6     instruction4
```

Enfin, il est possible d'enchaîner les conditions avec le mot clef `elif`, contraction de `else if`. Plusieurs conditions complémentaires peuvent être enchaînées. Avec ces instructions, il ne peut y avoir qu'un seul bloc exécuté : si une condition retourne vrai, le bloc associé est exécuté et est enchaîné avec l'instruction qui suit la structure conditionnelle, `instructionSuite` ci-dessous. Une instruction `else` peut être ajoutée à la fin mais n'est pas obligatoire.

```
1  if condition1:
2      instruction1 # bloc exécuté si condition1 vaut vrai
3      instruction2
4  elif condition2:
5      instruction3 # bloc exécuté si condition1 vaut faux et condition 2 vrai
6      instruction4
7  elif condition3:
8      instruction5 # bloc exécuté si condition1 et condition2 valent faux, et condition3 vrai
9      ...
10 else:
11     instructionN # bloc exécuté si aucune des conditions précédentes ne vaut vrai
12 instructionSuite
```



D Exercices d'application

1. Déterminer le résultat des expressions ci-dessous. Certaines expressions génèrent une erreur.

```
a, b = 3, 5
```

```
a < b
a < b and a != b
a > b or a == 3
a > b and a/0 > 5
a < b and a/0 > 5
```

2. Déterminer le type d'objet retourné par la fonction suivante et proposer une docstring. `n` doit être un entier. Rappel, l'opérateur `%` appliqué à des entiers retourne le reste de la division entière.

```
def mystere(n):
    r = n % 10
    return r == 0
```

3. Soit le script ci-dessous. Déterminer la valeur de `m` après son exécution, pour les différentes valeurs de `a` suivantes : 4, 5, 10, 15, 9

```
if a % 10 == 0:
    m = 10
elif a % 5 == 0:
    m = 5
elif a % 3 == 0:
    m = 3
else:
    m = 1
```

4. Écrire une fonction `mention(note)` ou `note` est un nombre (entier ou flottant) compris entre 0 et 20. La fonction retourne un entier représentatif de la mention : 1 si la note est strictement inférieure à 14, 2 pour une mention bien, 3 pour une mention très bien et 4 pour les félicitations

.

Corrigé