Collection Ressources Informatiques

Algorithmique

Techniques fondamentales de programmation

BTS, DUT Informatique

Exemples en Python

(nombreux exercices corrigés)

Franck EBEL
Sébastien ROHAUT

INFORMATIQUE TECHNIQUE

eni Editions

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence de l'ouvrage **RIPYALG** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 2
Les variables et opérateurs

Chapitre 3
Tests et logique booléenne

Chapitre 4
Les boucles

Chapitre 5
Les tableaux et structures

Chapitre 6
Les sous-programmes

Chapitre 7
Les fichiers

Chapitre 8
Notions avancées

Chapitre 9
Une approche de l'objet

Annexes

Corrigés des exercices

Chapitre 3
Tests et logique booléenne

1. Les tests et conditions

1.1 Principe

Dans le précédent chapitre vous avez pu vous familiariser avec les expressions mettant en place des opérateurs, qu'ils soient de calcul, de comparaison (l'égalité par exemple) ou booléens. Ces opérateurs et expressions trouvent tout leur sens une fois utilisés dans des conditions (qu'on appelle aussi des branchements conditionnels). Une expression évaluée est ou vraie (le résultat est différent de zéro) ou fausse. Suivant ce résultat, l'algorithme va effectuer une action, ou une autre. C'est le principe de la condition.

Grâce aux opérateurs booléens, l'expression peut être composée : plusieurs expressions sont liées entre elles à l'aide d'un opérateur booléen, éventuellement regroupées avec des parenthèses pour en modifier la priorité.

```
(a=1 OU (b*3=6)) ET c>10
```

est une expression tout à fait valable. Celle-ci sera vraie si chacun de ses composants respecte les conditions imposées. Cette expression est vraie si a vaut 1 et c est supérieur à 10 ou si b vaut 2 (2*3=6) et c est supérieur à 10.

Reprenez l'algorithme du précédent chapitre qui calcule les deux résultats possibles d'une équation du second degré. L'énoncé simplifié disait que pour des raisons pratiques seul le cas où l'équation a deux solutions fonctionne. Autrement dit, l'algorithme n'est pas faux dans ce cas de figure, mais il est incomplet. Il manque des conditions pour tester la valeur du déterminant : celui-ci est-il positif, négatif ou nul ? Et dans ces cas, que faire et comment le faire ?

Imaginez un second algorithme permettant de se rendre d'un point A à un point B. Vous n'allez pas le faire ici réellement, car c'est quelque chose de très complexe sur un réseau routier important. De nombreux sites Internet vous proposent d'établir un trajet avec des indications. C'est le résultat qui est intéressant. Les indications sont simples : allez tout droit, tournez à droite au prochain carrefour, faites trois kilomètres et au rond-point prenez la troisième sortie direction B. Dans la plupart des cas si vous suivez ce trajet vous arrivez à bon port. Mais quid des impondérables ? Par où allez-vous passer si la route à droite au prochain carrefour est devenue un sens interdit (ça arrive parfois, y compris avec un GPS, prudence) ou que des travaux empêchent de prendre la troisième sortie du rond-point ?

Reprenez le trajet : allez tout droit. Si au prochain carrefour la route à droite est en sens interdit : continuez tout droit puis prenez à droite au carrefour suivant puis à gauche sur deux kilomètres jusqu'au rond-point. Sinon : tournez à droite et faites trois kilomètres jusqu'au rond-point. Au rond-point, si la sortie vers B est libre, prenez cette sortie. Sinon, prenez vers C puis trois cents mètres plus loin tournez à droite vers B.

Ce petit parcours ne met pas uniquement en lumière la complexité d'un trajet en cas de détour, mais aussi les nombreuses conditions qui permettent d'établir un trajet en cas de problème. Si vous en possédez, certains logiciels de navigation par GPS disposent de possibilités d'itinéraire Bis, de trajectoire d'évitement sur telle section, ou encore pour éviter les sections à péage. Pouvez-vous maintenant imaginer le nombre d'expressions à évaluer dans tous ces cas de figure, en plus de la vitesse de chaque route pour optimiser l'heure d'arrivée ?

1.2 Que tester ?

Les opérateurs s'appliquent sur quasiment tous les types de données, y compris les chaînes de caractères, tout au moins en pseudo-code algorithmique (il faudra souvent utiliser des instructions spéciales du langage de programmation pour comparer des chaînes de caractères). Vous pouvez donc quasiment tout tester. Par tester, comprenez ici évaluer une expression qui est une condition. Une condition est donc le fait d'effectuer des tests pour, en fonction du résultat de ceux-ci, effectuer certaines actions ou d'autres.

Une condition est donc une affirmation : l'algorithme et le programme ensuite détermineront si celle-ci est vraie, ou fausse.

Une condition retournant VRAI ou FAUX a comme résultat un **booléen**.

Une condition est souvent une comparaison. Pour rappel, une comparaison est une expression composée de trois éléments :

– une première valeur : variable ou scalaire.
– un opérateur de comparaison.
– une seconde valeur : variable ou scalaire.

Les opérateurs de comparaison sont :

– L'égalité : **=**
– La différence : **!=** ou **<>**
– Inférieur : **<**
– Inférieur ou égal : **<=**
– Supérieur : **>**
– Supérieur ou égal : **>=**

Il est aussi possible que la condition soit une expression unaire : une valeur avec ou non un opérateur. Une variable pouvant être vraie ou fausse, elle peut donc suffire à évaluer la condition donnée.

Le pseudo-code algorithmique n'interdit pas de comparer des chaînes de caractères. Vous prendrez soin de ne comparer que les variables de types compatibles. Dans une condition une expression, quel que soit le résultat de celle-ci, sera toujours évaluée comme étant soit vraie, soit fausse.

Remarque

L'opérateur d'affectation peut aussi être utilisé dans une condition. Dans ce cas si vous affectez 0 à une variable, l'expression sera fausse, et si vous affectez n'importe quelle autre valeur, elle sera vraie.

En langage courant, il vous arrive de dire "choisissez un nombre entre 1 et 10". En mathématique, vous écrivez cela comme ceci :

```
1 ≤ nombre ≤ 10
```

Si vous écrivez ceci dans votre algorithme, attendez-vous à des résultats surprenants le jour où vous allez le convertir en véritable programme. En effet les opérateurs de comparaison ont une priorité, ce que vous savez déjà, mais l'expression qu'ils composent est aussi souvent évaluée de gauche à droite. Si la variable nombre contient la valeur 15, voici ce qui ce passe :

– L'expression 1 ≤ 15 est évaluée : elle est vraie.
– Et ensuite ? Tout va dépendre du langage de programmation utilisé, mais pour la plupart l'expression vrai ≤ 10 est vraie : "vrai" est ici le résultat de l'expression 1 ≤ 15. Vrai vaut généralement 1. Donc 1 ≤ 10 est vraie, ce n'est pas le résultat attendu.
– Le résultat est épouvantable : le programme considère l'expression comme étant vraie et le code correspondant va être exécuté.

Ce n'est probablement pas ce que vous attendiez. Vous devez donc proscrire cette forme d'expression.

Voici celles qui conviennent dans ce cas :

```
nombre>=1 ET nombre<=10
```

Ou encore

```
1<=nombre ET nombre<=10
```

1.3 Tests SI

1.3.1 Forme simple

Il n'y a, en algorithmique, qu'une seule instruction de test : "**Si**", qui prend cependant deux formes : une simple et une complexe. Le test SI permet d'exécuter un bloc d'instructions si la condition (la ou les expressions qui la composent) est vraie. La forme simple est la suivante :

```
Si booléen Alors
  Bloc d'instructions
FinSi
```

Notez ici que le booléen est la condition (ou expression). Comme indiqué précédemment, la condition peut aussi être représentée par une seule variable. Si elle contient 0, elle représente le booléen FAUX, sinon le booléen VRAI.

Que se passe-t-il si la condition est vraie ? Le bloc d'instructions situé après le "**Alors**" est exécuté. Sa taille (le nombre d'instructions) n'a aucune importance : de une ligne à n lignes, sans limite. Dans le cas contraire, le programme continue à l'instruction suivant le "**FinSi**". L'exemple suivant montre comment obtenir la valeur absolue d'un nombre avec cette méthode.

```
PROGRAMME ABS
VAR
  Nombre :entier
DEBUT
  nombre←-15
  Si nombre<0 Alors
    nombre←-nombre
  FinSi
  Afficher nombre
FIN
```

En Python, c'est le "if" qui doit être utilisé avec l'expression booléenne entre parenthèses. La syntaxe est celle-ci :

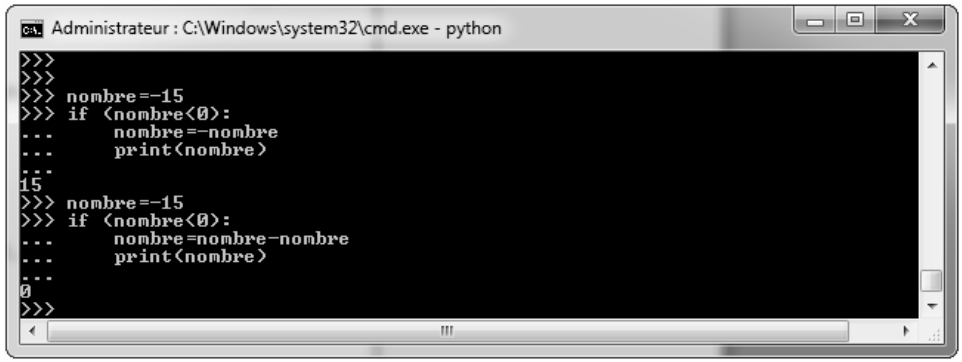
```
if(boolean):
/*code */
```

Une notion importante ici est que Python, contrairement a beaucoup de langages tels que le C ou le Java, n'utilise pas les accolades pour déterminer un bloc d'instruction mais l'indentation.

Si vous regardez le code ci-dessus, nous avons le test if avec la condition qui se termine par ":".

La ligne du dessous est indentée, c'est-à-dire que nous avons appuyé sur la touche [Tabulation] avant de commencer la ligne. Tout le code qui sera tabulé en dessous appartiendra au if.

```
nombre=-15
if(nombre<0):
   nombre=-nombre
   print(nombre)
```



Une condition avec une valeur booléenne se fait de la même manière. Le code suivant n'a qu'un rôle pédagogique : il met en place un drapeau ou flag permettant d'indiquer si la condition est vérifiée. Si le drapeau est vrai, alors le nombre est négatif. Le drapeau est ensuite testé pour afficher le résultat opposé.

```
PROGRAMME ABS
VAR
  nombre :entier

  drapeau :booleen
DEBUT

  drapeau←FAUX
  nombre←-15
  Si nombre<0 Alors
    drapeau←VRAI
  FinSi
```