

Algorithmique

Techniques fondamentales
de programmation

BTS,
DUT Informatique

Exemples en Python
(nombreux exercices corrigés)

Franck EBEL
Sébastien ROHAUT

Téléchargement
www.editions-eni.fr



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.

Avant-propos

Chapitre 1 Introduction à l'algorithmique

- 1. Les fondements de l'informatique 13
 - 1.1 Architecture de Von Neumann 13
 - 1.2 La machine de Turing 17
 - 1.3 Représentation interne des instructions et des données 19
 - 1.3.1 Le binaire 19
 - 1.3.2 Les octets et les mots 22
 - 1.3.3 L'hexadécimal 23
- 2. L'algorithmique 24
 - 2.1 Programmer, c'est un art 24
 - 2.2 Définition : L'algorithme est une recette 26
 - 2.3 Pourquoi utiliser un algorithme ? 27
 - 2.4 Le formalisme 28
 - 2.4.1 Les algorigrammes 29
 - 2.4.2 L'algorithme sous forme de texte 30
 - 2.5 La complexité 32
 - 2.6 Les structures algorithmiques 35
- 3. Les langages d'implémentation 36
 - 3.1 Quel langage ? 36
 - 3.2 Classifications des langages 39
 - 3.2.1 Haut niveau, bas niveau 39
 - 3.2.2 Diverses classifications 40
 - 3.2.3 Compilé ou interprété 41

2 Algorithmique

Techniques fondamentales de programmation (exemples en Python)

3.3	La machine virtuelle	42
3.4	Python	44
3.4.1	Les avantages	44
3.4.2	Un premier programme Python	47
4.	Exercices	48

Chapitre 2

Les variables et opérateurs

1.	La variable	51
1.1	Principe	51
1.2	Déclaration	54
1.3	Les types	54
1.3.1	Les nombres	56
1.3.2	Autres types numériques	58
1.3.3	Les caractères	59
1.3.4	Le type booléen	61
1.4	Affectation	63
1.4.1	Affectation de valeurs	63
1.4.2	Affectation de variables	66
1.5	Saisie et affichage	67
1.5.1	La fonction print()	68
1.5.2	La fonction input()	69
1.6	Les constantes	70
2.	Opérateurs et calculs	71
2.1	Les affectations	71
2.2	Les opérateurs arithmétiques	71
2.3	Les opérateurs booléens	76
2.4	Les opérateurs de comparaison	79
2.4.1	L'égalité	80
2.4.2	La différence	81
2.4.3	Inférieur, supérieur	82
2.5	Le cas des chaînes de caractères	83

2.6	La précedence des opérateurs	84
3.	Pour aller plus loin	85
3.1	Les nombres négatifs	85
3.2	La représentation des nombres réels	87
3.3	Les dates	92
3.4	Les caractères	93
4.	Types et langages	95
4.1	Langages typés ou non	95
4.2	La gestion de la mémoire	96
5.	Les types spécifiques à Python.	97
5.1	Les listes	97
5.2	Les tuples.	99
5.3	Le type set	101
5.4	Les dictionnaires	102
6.	Exercices	103

Chapitre 3
Tests et logique booléenne

1.	Les tests et conditions	107
1.1	Principe	107
1.2	Que tester ?	109
1.3	Tests SI	111
1.3.1	Forme simple	111
1.3.2	Forme complexe.	113
1.4	Tests imbriqués.	116
1.5	Choix multiples	120
1.6	Des exemples complets	122
1.6.1	Le lendemain d’une date	122
1.6.2	La validité d’une date	126
1.6.3	L’heure dans n secondes	127

4 Algorithmique

Techniques fondamentales de programmation (exemples en Python)

2. L'algèbre booléen	131
2.1 L'origine des tests	131
2.2 Petites erreurs, grosses conséquences	132
2.2.1 Ariane 5	133
2.2.2 Mars Climate Orbiter	133
2.3 George Boole	134
2.4 L'algèbre	135
2.4.1 Établir une communication	135
2.4.2 La vérité	137
2.4.3 La loi ET	137
2.4.4 La loi OU	138
2.4.5 Le contraire	139
2.4.6 Les propriétés	139
2.4.7 Quelques fonctions logiques	143
2.4.8 Avec plus de deux variables	146
2.5 Une dernière précision	149
3. Exercices	150

Chapitre 4 Les boucles

1. Les structures itératives	153
1.1 Définition	153
1.2 Quelques usages simples	154
2. Tant Que	155
2.1 Structure générale	155
2.2 Boucles infinies et "break"	156
2.3 Des exemples	158
2.3.1 Une table de multiplication	158
2.3.2 Une factorielle	160
2.3.3 x à la puissance y	161
2.3.4 Toutes les tables de multiplication	163
2.3.5 Saisie de notes et calcul de moyennes	165

2.3.6	Rendez la monnaie	172
2.3.7	Trois boucles	176
3.	Répéter ... Jusqu'à	178
3.1	Différences fondamentales	178
3.2	Quelques exemples adaptés	180
3.2.1	La factorielle	180
3.2.2	Les trois boucles	180
4.	Pour ... Fin Pour	181
4.1	Une structure pour compter.....	181
4.2	... mais pas indispensable	182
4.3	Quelle structure choisir ?	182
4.4	Un piège à éviter	183
4.5	Quelques exemples	184
4.5.1	De nouveau trois boucles	184
4.5.2	La factorielle	186
4.5.3	Racine carrée avec précision	187
4.5.4	Calcul du nombre PI	189
5.	Exercices	192

Chapitre 5

Les tableaux et structures

1.	Présentation	195
1.1	Principe et définition	195
1.1.1	Simplifier les variables	195
1.1.2	Les dimensions	197
1.1.3	Les types	198
1.1.4	Déclaration	199
1.1.5	Utilisation	200
1.1.6	Les tableaux dynamiques	200
1.2	Python et les tableaux	202

6 Algorithmique

Techniques fondamentales de programmation (exemples en Python)

1.3	Représentation en mémoire	206
1.3.1	Représentation linéaire	206
1.3.2	Représentation par référence	208
2.	Manipulations simples	211
2.1	Recherche d'un élément	211
2.2	Le plus grand/petit, la moyenne	214
2.3	Le morpion	216
3.	Algorithmes avancés	220
3.1	Les algorithmes des tris	220
3.1.1	Le principe	220
3.1.2	Le tri par création	221
3.1.3	Le tri par sélection	222
3.1.4	Le tri à bulles	224
3.1.5	Le tri par insertion	229
3.1.6	Le tri Shell	232
3.2	Recherche par dichotomie	234
4.	Structures et enregistrements	237
4.1	Principe	237
4.2	Déclaration	238
4.2.1	Type structuré	238
4.2.2	Enregistrement	239
4.3	Utiliser les enregistrements	241
4.3.1	Utiliser les champs	241
4.3.2	Un enregistrement dans une structure	243
4.3.3	Un tableau dans une structure	244
4.4	Les tableaux d'enregistrements	246
4.4.1	Les tables	246
4.4.2	Une table comme champ	247
4.5	Et Python ?	248
5.	Exercices	249

Chapitre 6
Les sous-programmes

- 1. Présentation 251
 - 1.1 Principe 251
 - 1.2 Déclaration et définition 253
 - 1.2.1 Dans un algorithme. 253
 - 1.2.2 En Python. 255
 - 1.3 Appel 256
 - 1.4 Fonctions et procédures 258
 - 1.4.1 Les procédures 258
 - 1.4.2 Les fonctions 259
 - 1.5 Variables locales et globales 261
 - 1.5.1 Locales 261
 - 1.5.2 Globales 262
 - 1.5.3 Variables globales et Python 264
 - 1.6 Les paramètres 264
 - 1.6.1 Les procédures 265
 - 1.6.2 Les fonctions 267
 - 1.6.3 Paramètres et Python 269
 - 1.6.4 Petite application fonctionnelle 272
 - 1.7 Sous-programmes prédéfinis 274
 - 1.7.1 Un choix important. 274
 - 1.7.2 Quelques exemples 275
 - 1.8 Dernier cas : les tableaux 282
- 2. Les sous-programmes récursifs 285
 - 2.1 Principe 285
 - 2.2 Un premier exemple : la factorielle 286
 - 2.3 Un exemple pratique : les tours de Hanoï. 289
- 3. Exercices 291

Chapitre 7 Les fichiers

1. Les différents fichiers	293
1.1 Préambule	293
1.2 Problématique	294
1.3 Définition	295
1.4 Les formats	295
1.4.1 Types de contenus	295
1.4.2 Le fichier binaire	297
1.4.3 Le fichier texte	298
1.4.4 Quel format utiliser ?	300
1.5 Les accès aux fichiers	301
1.5.1 Séquentiel	301
1.5.2 Accès direct	302
1.5.3 Indexé	302
1.5.4 Autre ?	302
2. Les enregistrements	303
2.1 Les délimiteurs	303
2.2 Largeur fixe	306
2.3 Principes d'accès	307
2.3.1 Étapes de base	307
2.3.2 Identificateurs de fichiers et canaux	308
2.3.3 Les modes d'ouverture	310
3. Fichier texte séquentiel	311
3.1 Ouvrir et fermer un fichier	311
3.2 Lire et écrire des enregistrements	312
3.2.1 Lecture	312
3.2.2 Écriture	314
3.3 Les enregistrements structurés	318
3.4 Exemple en Python	320

- 4. Les fichiers binaires 323
 - 4.1 Nouvelles instructions 323
 - 4.2 Exemple 323
- 5. Exercices 325

Chapitre 8
Notions avancées

- 1. Les pointeurs et références 327
 - 1.1 Rappels sur la mémoire et les données 327
 - 1.1.1 Structure de la mémoire 327
 - 1.1.2 Python : des limites qui n'en sont pas 329
 - 1.1.3 Brefs exemples en C 330
 - 1.2 Le pointeur 330
 - 1.2.1 Principe et définition 330
 - 1.2.2 Le C roi des pointeurs 332
 - 1.2.3 Applications 333
 - 1.3 Notation algorithmique 336
 - 1.3.1 Déclarer et utiliser les pointeurs 336
 - 1.3.2 Allocation dynamique 339
 - 1.4 Python et les références 341
 - 1.4.1 Différences entre le C et Python 341
 - 1.4.2 Références sur les objets 341
 - 1.4.3 Les types primitifs 344
 - 1.4.4 Références sur structures 344
 - 1.4.5 Le piège en Python 346
- 2. Les listes chaînées 347
 - 2.1 Listes chaînées simples 347
 - 2.1.1 Principe 347
 - 2.1.2 Création 351
 - 2.1.3 Parcours de la liste 353
 - 2.1.4 Recherche 353
 - 2.1.5 Ajout d'un élément 355

10 Algorithmique

Techniques fondamentales de programmation (exemples en Python)

2.1.6	Suppression d'un élément	358
2.1.7	Supprimer toute la liste	361
2.1.8	Parcours récursif	362
2.2	L'implémentation en Python	362
2.3	Autres exemples de listes	366
2.3.1	Listes circulaires	366
2.3.2	Listes d'éléments triés	366
2.3.3	Listes doublement chaînées	366
2.3.4	Files et piles	367
3.	Les arbres	368
3.1	Principe	368
3.2	Définitions	370
3.2.1	Base	370
3.2.2	Terminologie	370
3.2.3	Description horizontale	371
3.2.4	Description verticale	371
3.2.5	L'arbre binaire	371
3.3	Parcours d'un arbre	372
3.4	Arbre binaire ordonné	375
3.4.1	Principe	375
3.4.2	Recherche d'un élément	375
3.4.3	Ajout d'un élément	377
3.4.4	Suppression d'un nœud	378
4.	Exercices	379

Chapitre 9
Une approche de l'objet

- 1. Principe de l'objet, une notion évidente 381
 - 1.1 Avant de continuer 381
 - 1.2 Rappels sur la programmation procédurale 382
 - 1.2.1 Les données 382
 - 1.2.2 Les traitements 383
 - 1.3 L'objet 383
 - 1.3.1 Dans la vie courante 383
 - 1.3.2 En informatique 385
 - 1.4 Classe, objets 389
 - 1.5 Déclaration et accès 390
 - 1.6 Les méthodes 392
 - 1.7 Portée des membres 393
 - 1.8 Encapsulation des données 395
 - 1.9 L'héritage 397
 - 1.9.1 Principe 397
 - 1.9.2 Commerce 399
 - 1.9.3 Hiérarchie 400
 - 1.9.4 Simple ou multiple 401
 - 1.10 Le polymorphisme 402
 - 1.10.1 Principe 402
 - 1.10.2 Le polymorphisme ad hoc 402
 - 1.10.3 Le polymorphisme d'héritage 403
 - 1.10.4 Le polymorphisme paramétrique 405
- 2. Manipuler les objets 406
 - 2.1 Les constructeurs 406
 - 2.1.1 Déclaration 406
 - 2.1.2 Appel implicite 407
 - 2.1.3 L'héritage 409
 - 2.2 Les destructeurs 411
 - 2.3 Les membres statiques 412
 - 2.4 Classes et méthodes abstraites 414

12 Algorithmique

Techniques fondamentales de programmation (exemples en Python)

2.5 Interfaces	417
3. L'objet en Python	419
3.1 Les langages objet	419
3.2 Déclaration des classes et objets	420
3.3 Héritage	423
3.4 Interfaces	424
4. Exercices	427

Annexes

Corrigés des exercices

1. Introduction à l'algorithmique	429
2. Les variables et opérateurs	433
3. Tests et logique booléenne	440
4. Les boucles	448
5. Les tableaux et structures	462
6. Les sous-programmes	469
7. Les fichiers	475
8. Notions avancées	481
9. Une approche de l'objet	484

Index	493
-----------------	-----

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 :

$$1 \leq \text{nombre} \leq 10$$

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 se passe :

- L'expression $1 \leq 15$ est évaluée : elle est vraie.
- Et ensuite ? Tout va dépendre du langage de programmation utilisé, mais pour la plupart l'expression $\text{vrai} \leq 10$ est vraie : "vrai" est ici le résultat de l'expression $1 \leq 15$. Vrai vaut généralement 1. Donc $1 \leq 10$ 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 */
```

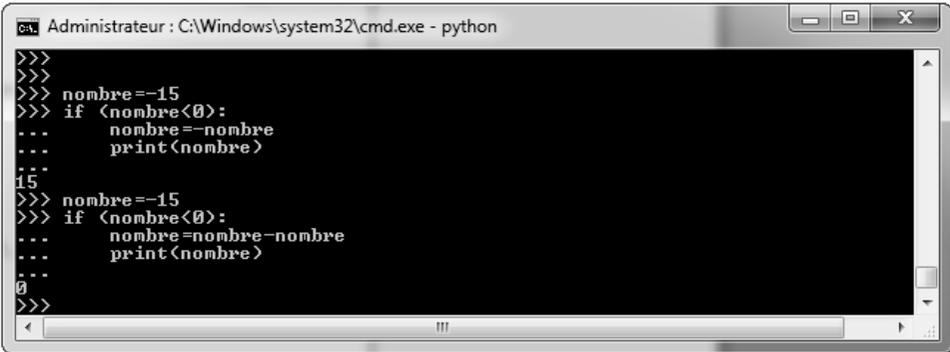
Une notion importante ici est que Python, contrairement à beaucoup de langages tels que le C ou le Java, n'utilise pas les accolades pour déterminer un bloc d'instruction mais l'indentation.

Techniques fondamentales de programmation (exemples en Python)

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)
```



```
Administrateur : C:\Windows\system32\cmd.exe - python
>>>
>>>
>>> nombre=-15
>>> if (nombre<0):
...     nombre=-nombre
...     print(nombre)
...
15
>>> nombre=-15
>>> if (nombre<0):
...     nombre=nombre-nombre
...     print(nombre)
...
0
>>>
```

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
```