

# 1 Variables et affectations

## Les variables en algorithmique

- Les variables algorithmiques peuvent servir à stocker des données de différents types, mais nous nous contenterons ici d'utiliser des variables du type NOMBRE.
- La valeur d'une variable peut changer au fil des instructions de l'algorithme.
- Les opérations sur les variables s'effectuent ligne après ligne et les unes après les autres.
- Quand l'ordinateur exécute une ligne du type mavariable, il effectue d'abord le calcul et stocke ensuite le résultat dans mavariable.

### ► Activité n°1

On considère l'algorithme suivant :

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: z EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:   x PREND_LA_VALEUR 2
7:   y PREND_LA_VALEUR 3
8:   z PREND_LA_VALEUR x+y
9: FIN_ALGORITHME
    
```

Après exécution de l'algorithme ; la variable x contient la valeur  ; la variable y contient la valeur  et la variable z contient la valeur .

### ► Activité n°2

On considère l'algorithme suivant :

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   x PREND_LA_VALEUR 2
5:   x PREND_LA_VALEUR x+1
6: FIN_ALGORITHME
    
```

Après exécution de l'algorithme, la variable x contient la valeur :

### ► Activité n°3

Ajoutons la ligne « x PREND\_LA\_VALEUR 4\*x » à la fin du code précédent. Ce qui donne :

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   x PREND_LA_VALEUR 2
5:   x PREND_LA_VALEUR x+1
6:   x PREND_LA_VALEUR 4*x
7: FIN_ALGORITHME
    
```

Après exécution de l'algorithme, la variable x contient la valeur :

### ► Activité n°4

On considère l'algorithme suivant :

```

1: VARIABLES
2: A EST_DU_TYPE NOMBRE
3: B EST_DU_TYPE NOMBRE
4: C EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:   A PREND_LA_VALEUR 5
7:   B PREND_LA_VALEUR 3
8:   C PREND_LA_VALEUR A+B
9:   B PREND_LA_VALEUR B+A
10:  A PREND_LA_VALEUR C
11: FIN_ALGORITHME
    
```

Après exécution de l'algorithme ; la variable A contient la valeur  ; la variable B contient la valeur  et la variable C contient la valeur .

### ► Activité n°5

On considère l'algorithme suivant :

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: z EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:   LIRE x
7:   y PREND_LA_VALEUR x-2
8:   z PREND_LA_VALEUR -3*y-4
9:   AFFICHER z
10: FIN_ALGORITHME
    
```

On cherche maintenant à obtenir un algorithme équivalent sans utiliser la variable y. Compléter la ligne 6 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: z EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   LIRE x
6:   z PREND_LA_VALEUR .....
7:   AFFICHER z
8: FIN_ALGORITHME
    
```

## 2 Instructions conditionnelles

### — SI...ALORS...SINON —

Comme nous l'avons vu ci-dessus, un algorithme permet d'exécuter une liste d'instructions les unes à la suite des autres. Mais on peut aussi "dire" à un algorithme de n'exécuter des instructions que si une certaine condition est remplie. Cela se fait grâce à la commande SI...ALORS :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
```

Il est aussi possible d'indiquer en plus à l'algorithme de traiter le cas où la condition n'est pas vérifiée. On obtient alors la structure suivante :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
SINON
  DEBUT_SINON
  ...
  FIN_SINON
```

### ► Activité n°6

On cherche à créer un algorithme qui demande un nombre à l'utilisateur et qui affiche la racine carrée de ce nombre s'il est positif. Compléter la ligne 6 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: racine EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   LIRE x
6:   SI (.....) ALORS
7:     DEBUT_SI
8:       racine PREND_LA_VALEUR sqrt(x)
9:       AFFICHER racine
10:    FIN_SI
11: FIN_ALGORITHME
```

### ► Activité n°7

On cherche à créer un algorithme qui demande à l'utilisateur d'entrer deux nombres (stockés dans les variables x et y) et qui affiche le plus grand des deux. Compléter les ligne 9 et 13 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   LIRE x
6:   LIRE y
7:   SI (x>y) ALORS
8:     DEBUT_SI
9:       AFFICHER .....
10:    FIN_SI
11:  SINON
12:    DEBUT_SINON
13:    AFFICHER .....
14:    FIN_SINON
15: FIN_ALGORITHME
```

### ► Activité n°8

On considère l'algorithme suivant :

```
1: VARIABLES
2: A EST_DU_TYPE NOMBRE
3: B EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   A PREND_LA_VALEUR 1
6:   B PREND_LA_VALEUR 3
7:   SI (A>0) ALORS
8:     DEBUT_SI
9:       A PREND_LA_VALEUR A+1
10:    FIN_SI
11:   SI (B>4) ALORS
12:     DEBUT_SI
13:       B PREND_LA_VALEUR B-1
14:     FIN_SI
15: FIN_ALGORITHME
```

Après exécution de l'algorithme :

– La variable A contient la valeur :

– La variable B contient la valeur :

### ► Activité n°9

On cherche à concevoir un algorithme correspondant au problème suivant :

- on demande à l'utilisateur d'entrer un nombre (représenté par la variable x)
- si le nombre entré est différent de 1, l'algorithme doit stocker dans une variable y la valeur de  $1/(x-1)$  et afficher la valeur de y (note : la condition x différent de 1 s'exprime avec le code  $x \neq 1$ ). On ne demande pas de traiter le cas contraire.

Compléter l'algorithme ci-dessous pour qu'il réponde au problème.

```

1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   LIRE .....
6:   SI (.....) ALORS
7:     DEBUT_SI
8:     ..... PREND_LA_VALEUR .....
9:     AFFICHER .....
10:    FIN_SI
11: FIN_ALGORITHME

```

### 3 Boucles

#### Boucles POUR...DE...A

- Les boucles permettent de répéter des instructions autant de fois que l'on souhaite.
- Lorsqu'on connaît par avance le nombre de fois que l'on veut répéter les instructions, on utilise une boucle du type POUR...DE...A dont la structure est la suivante :

```

POUR...ALLANT_DE...A...
  DEBUT_POUR
  ...
  FIN_POUR

```

- Exemple : l'algorithme ci-dessous permet d'afficher la racine carrée de tous les entiers de 1 jusqu'à 50.

```

1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: racine EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   POUR n ALLANT_DE 1 A 50
6:     DEBUT_POUR
7:     racine PREND_LA_VALEUR sqrt(n)
8:     AFFICHER racine
9:     FIN_POUR
10: FIN_ALGORITHME

```

La variable n est appelée « compteur de la boucle ».

- Remarques :

- La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
- Dans AlgoBox, cette variable est automatiquement augmentée de 1 à chaque fois.
- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre DEBUT\_POUR et FIN\_POUR ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur.

#### ► Activité n°10

On cherche à concevoir un algorithme qui affiche, grâce à une boucle POUR...DE...A, les résultats des calculs suivants :  $8*1$  ;  $8*2$  ;  $8*3$  ;  $8*4$  ; ... jusqu'à  $8*10$ .

La variable n sert de compteur à la boucle et la variable produit sert à stocker et afficher les résultats. Compléter les lignes 5 et 7 dans l'algorithme ci-dessous pour qu'il réponde au problème :

```

1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: produit EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   POUR n ALLANT_DE ... A ...
6:     DEBUT_POUR
7:     produit PREND_LA_VALEUR .....
8:     AFFICHER produit
9:     FIN_POUR
10: FIN_ALGORITHME

```

#### ► Activité n°11

On considère l'algorithme suivant :

```

1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: somme EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   somme PREND_LA_VALEUR 0
6:   POUR n ALLANT_DE 1 A 100
7:     DEBUT_POUR
8:     somme PREND_LA_VALEUR somme+n
9:     FIN_POUR
10:  AFFICHER somme
11: FIN_ALGORITHME

```

Compléter les phrases suivantes :

- Après exécution de la ligne 5, la variable somme contient la valeur :
- Lorsque le compteur n de la boucle vaut 1 et après exécution du calcul ligne 8, la variable somme vaut :
- Lorsque le compteur n de la boucle vaut 2 et après exécution du calcul ligne 8, la variable somme vaut :
- Lorsque le compteur n de la boucle vaut 3 et après exécution du calcul ligne 8, la variable somme vaut :

Que permet de calculer cet algorithme ?

#### ► Activité n°12

Compléter les lignes 6 et 8 de l'algorithme ci-dessous pour qu'il permette de calculer la somme  $5^2 + 6^2 + 7^2 + \dots + 24^2 + 25^2$ .

```

1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: somme EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   somme PREND_LA_VALEUR 0
6:   POUR n ALLANT_DE .... A .....
7:     DEBUT_POUR
8:     somme PREND_LA_VALEUR somme+.....
9:     FIN_POUR
10: AFFICHER somme
11: FIN_ALGORITHME

```

### Boucles TANT QUE...

- Il n'est pas toujours possible de connaître par avance le nombre de répétitions nécessaires à un calcul. Dans ce cas là, il est possible d'avoir recours à la structure TANT QUE... qui se présente de la façon suivante :

```

TANT_QUE...FAIRE
  DEBUT_TANT_QUE
  ...
  FIN_TANT_QUE

```

Cette structure de boucle permet de répéter une série d'instructions (comprises entre DEBUT\_TANT\_QUE et FIN\_TANT\_QUE) tant qu'une certaine condition est vérifiée.

- Exemple : Comment savoir ce qu'il reste si on enlève 25 autant de fois que l'on peut au nombre 583? Pour cela on utilise une variable n, qui contient 583 au début, à laquelle on enlève 25 tant que c'est possible, c'est à dire tant que n est supérieur ou égal à 25.

```

1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   n PREND_LA_VALEUR 583
5:   TANT_QUE (n>=25) FAIRE
6:     DEBUT_TANT_QUE
7:     n PREND_LA_VALEUR n-25
8:     FIN_TANT_QUE
9:   AFFICHER n
10: FIN_ALGORITHME

```

### Remarques :

- Si la condition du TANT QUE... est fautive dès le début, les instructions entre DEBUT\_TANT\_QUE et FIN\_TANT\_QUE ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).
- Il est indispensable de s'assurer que la condition du TANT QUE... finisse par être vérifiée (le code entre DEBUT\_TANT\_QUE et FIN\_TANT\_QUE doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme ne pourra pas fonctionner.

### ► Activité n°13

On cherche à connaître le plus petit entier N tel que  $2^N$  soit supérieur ou égal à 10000. Pour résoudre ce problème de façon algorithmique :

- On utilise une variable N à laquelle on donne au début la valeur 1.
  - On augmente de 1 la valeur de N tant que  $2^N$  n'est pas supérieur ou égal à 10000.
- Une structure TANT QUE est particulièrement adaptée à ce genre de problème car on ne sait pas a priori combien de calculs seront nécessaires.

Compléter les lignes 5 et 7 de l'algorithme ci-dessous pour qu'il réponde au problème :

(remarque :  $\text{pow}(2, N)$  est le code AlgoBox pour calculer  $2^N$ )

```

1: VARIABLES
2: N EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   N PREND_LA_VALEUR 1
5:   TANT_QUE (pow(2,N).....) FAIRE
6:     DEBUT_TANT_QUE
7:     N PREND_LA_VALEUR .....
8:     FIN_TANT_QUE
9:   AFFICHER N
10: FIN_ALGORITHME

```

### ► Activité n°14

On considère le problème suivant :

- On lance une balle d'une hauteur initiale de 300 cm.
- On suppose qu'à chaque rebond, la balle perd 10% de sa hauteur (la hauteur est donc multipliée par 0.9 à chaque rebond).
- On cherche à savoir le nombre de rebonds nécessaire pour que la hauteur de la balle soit inférieure ou égale à 10 cm.

Compléter les lignes 7 et 10 de l'algorithme ci-dessous pour qu'il réponde au problème.

```

1: VARIABLES
2: nombre_rebonds EST_DU_TYPE NOMBRE
3: hauteur EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   nombre_rebonds PREND_LA_VALEUR 0
6:   hauteur PREND_LA_VALEUR 300
7:   TANT_QUE (hauteur.....) FAIRE
8:     DEBUT_TANT_QUE
9:     nombre_rebonds PREND_LA_VALEUR nombre_rebonds+1
10:    hauteur PREND_LA_VALEUR .....
11:    FIN_TANT_QUE
12:  AFFICHER nombre_rebonds
13: FIN_ALGORITHME

```