Les sous-programmes (VBA)
Corrigé

Résumé

Ce document décrit les sous-programmes, outil essentiel pour structurer un programme car il permet au programmeur de définir ses propres instructions et opérateurs.

Table des matières

1 Les sous-programmes en VBA

En Visual BASIC, les notions de procédures et fonctions sont a peu près équivalentes à leur sens algorithmique. Les procédures sont déclarées dans une section **`Sub ... End Sub`** tandis que les fonctions utilisent une section **`Function ... End Function`**.

2 Procédures (Sub ... End Sub)

2.1 Procédure sans paramètre (ou macro)

A la base, une procédure est un ensemble d'instruction auquel on a donné un nom (instruction composée). Il faut bien distinguer la **définition** d'une procédure, qui décrit la liste des instructions qui la composent, des **appels** à la procédure, chacun d'entre eux (au sein d'une autre procédure ou fonction) déclenchant son exécution, c'est-à-dire l'exécution des instructions qui la composent.

Voyons comment cela s'écrit en Visual Basic dans le cas du robot, en commençant par les 2 procédures élémentaires AVANCER et PIVOTER, où l'on va simuler le déplacement du robot simplement en affichant le message correspondant à l'écran :

```
1 Sub AVANCER()
2 'ROLE : Simuler l'avance du robot en affichant le message AVANCER
3     Afficher "AVANCER"
4 End Sub
5
6 Sub PIVOTER()
7 'ROLE : Simuler le pivotement du robot en affichant le message PIVOTER A DROITE
8     Afficher "PIVOTER A DROITE"
9 End Sub
```

Remarque : Noter la présence d'un commentaire commençant par le mot ROLE, que l'un utilisera **systématiquement** pour décrire brièvement mais précisément le rôle de la procédure.
En Visual BASIC, une procédure sans paramètre est appelée **Macro**. Seules, des macros peuvent être associées à des boutons placés dans une feuille de calcul.

2.2 Procédure avec variable locale et paramètre d'entrée (ByVal)

Les 2 procédures élémentaires AVANCER et PIVOTER vont être utilisées (ou appelées) par les 3 procédures `Tourner_droite`, `Tourner_gauche` et `Progresser`, assurant les mouvements de base :

```
1 Sub Tourner_droite()
2 'ROLE : Faire tourner le robot à droite
3     PIVOTER
4 End Sub
5
6 Sub Tourner_gauche()
7 'ROLE : Faire tourner le robot à gauche
8     PIVOTER
```

```
    PIVOTER
    PIVOTER
End Sub

Sub Progresser(ByVal n As Integer)
'ROLE : Faire avancer le robot de n cases dans la direction courante.
'ENTREE n (doit être strictement positif)
    Dim i As Integer ' pour compter le nombre de fois où le robot avance
    For i = 1 To n
        AVANCER
    Next i
End Sub
```

Les procédures `Tourner_droite` et `Tourner_gauche` contiennent des appels de la procédure `PIVOTER` : une fois depuis `Tourner_droite` et 3 fois depuis `Tourner_gauche`.

La procédure `Progresser` illustre l'utilisation d'une **variable locale** i et d'un **paramètre d'entrée** n, information dont la connaissance préalable de la valeur est indispensable à la bonne exécution de la procédure lors de chaque appel. La variable locale i sera créée en début d'exécution de la procédure, et détruite en fin d'exécution.

Le nom est le type du paramètre formel doivent être spécifié dans les parenthèses suivant le nom de la procédure, au sein de la ligne d'en-tête. Le mot-clé **`ByVal`** spécifie un **passage " par valeur "**, que l'on peut aussi qualifier de " par copie " : ceci signifie que, lors d'un appel à la procédure, le paramètre formel se comporte comme une variable locale (créée au moment de l'appel) initialisée implicitement avec la valeur du paramètre effectif spécifié dans l'instruction d'appel (cf. ci-dessous dans la procédure `GuiderRobot` : par exemple `Progresser 2`, pour indiquer une avancée de 2 cases) : cette variable locale se comporte donc comme une **copie** du paramètre effectif, initialisé avec la **valeur** de ce dernier.

Ce mode de passage est exclusivement réservé à des paramètres d'entrée, et ne pourra pas être utilisé pour des paramètres de sortie ou de mise à jour.

Le mode de passage algorithmique (in, out, in out) sera **obligatoirement** précisé sous forme de commentaire juste après le rôle de la procédure. Pour un paramètre d'entrée, on utilisera le mot ENTREE en majuscule, pour mieux le mettre en valeur. Tout comme pour le rôle de la procédure, cette obligation n'est pas d'ordre syntaxique (puisqu'il s'agit d'un commentaire) mais d'ordre méthodologique. L'ensemble de la ligne d'en-tête, du rôle et des modes de passage des paramètres constitue les **spécifications** de la procédure. La lecture de ces spécifications doit être suffisante pour comprendre ce que fait la procédure et l'appeler de manière correcte sans avoir besoin d'aller voir les instructions qui la composent.

La procédure principale de déplacement du robot s'écrira donc comme suit :

```
Sub GuiderRobot()
'ROLE : Sortir de la salle de cours et aller jusqu'au secrétariat

   '1. Sortir de la salle de cours
    Progresser 2
    Tourner_gauche
    Progresser 3

   '2. Longer le couloir (jusqu'à la porte du vestibule)
```

```
10      Tourner_droite
11      Progresser 9
12
13     '3. Traverser le vestibule (et entrer dans le secrétariat)
14      Tourner_droite
15      Progresser 3
16      Tourner_droite
17      Progresser 1
18      Tourner_gauche
19      Progresser 1
20
21  End Sub
```

Attention : En Visual BASIC, contrairement à la notation algorithmique et à la plupart des autres langages, **les paramètres effectifs spécifiés dans chaque appel de procédure ne doivent pas être mis entre parenthèses**.
Nous verrons plus loin comment utiliser plusieurs paramètres, et comment passer des paramètres de sortie ou de mise à jour.
Remarque : En Visual BASIC, l'ordre de définition des procédures n'a pas d'influence sur l'exécution du programme ; en particulier, il n'est pas nécessaire de définir une procédure avant une procédure qui l'appelle. L'ordre de déclaration doit simplement respecter un ordre logique guidé par la clarté et la lisibilité du programme.

3 Fonctions (Function ... End Function)

3.1 Fonction à 1 paramètre d'entrée

Une fonction est un groupe d'instructions auquel on a donné un nom et qui renvoie une valeur (expression composée). Tout comme pour une procédure, il faut distinguer la définition (unique) de la fonction, et les appels (multiples) de la fonction par une autre procédure ou fonction. Voici la définition de la fonction `Carre`, suivie de la définition de la procédure `TesterCarre` qui appelle la fonction `Carre` :

```
1  Function Carre(ByVal x As Double) As Double
2  'ROLE : renvoyer le carré d'un nombre réel
3      Carre = x * x
4  End Function
5
6  Sub TesterCarre()
7  'ROLE : tester la fonction carre
8      Dim Nombre As Double
9      Afficher "Saisir␣un␣nombre␣:␣"
10     Saisir Nombre, 16
11     Afficher "Son␣carré␣vaut␣" & Carre(Nombre)
12 End Sub
```

Deux points différencient la définition d'une fonction par rapport à une procédure :

– Le type de la fonction doit obligatoirement conclure la ligne d'en-tête, après la liste de paramètres
–
– Le renvoie du résultat de la fonction s'effectue sous forme d'une affectation à son nom
–

Une fonction est appelable en tant qu'expression, c'est-à-dire partout où une valeur est autorisée :

– en partie droite d'une affectation : `r = Carre (2)`
–
– au sein d'une condition (expression booléenne) : `If Carre(r) < 2 Then`
–
– comme paramètre effectif d'entrée dans l'appel d'une autre procédure ou fonction : `Afficher Carre (2)`
–

Remarque : Lors d'un appel de fonction, les paramètres effectifs doivent être mis entre parenthèses, à l'instar de la convention universellement utilisée pour la notation algorithmique et la plupart des langages informatiques.

3.2 Fonction à 2 paramètres d'entrée

On peut déclarer plusieurs paramètres formels dans la définition d'une fonction, en respectant la même contrainte que pour des déclarations de variables : chaque paramètre formel doit obligatoirement être suivi de son type, même s'il y a plusieurs paramètres de même type.

```
1  Function Max(ByVal a As Integer, ByVal b As Integer) As Integer
2  'ROLE : renvoyer le plus grand de deux nombres entiers
3  'ENTREE a, b
4      If a > b Then Max = a Else Max = b
5  End Function
6
7  Sub TesterMax()
8  'ROLE : tester la fonction Max
9      Dim Nombre1 As Integer, Nombre2 As Integer
10     Afficher "Saisir␣un␣premier␣nombre␣:␣"
11     Saisir Nombre1, 2
12     Afficher "Saisir␣un␣second␣nombre␣:␣"
13     Saisir Nombre2, 5
14     Afficher "Le␣plus␣grand␣des␣deux␣est␣" & Max(Nombre1, Nombre2)
15 End Sub
```

Bien évidemment, l'appel de la fonction doit comporter autant de paramètres effectifs que la définition en a spécifiés, en respectant leur ordre.

4 Procédures avec paramètres de sortie ou de mise à jour (ByRef)

L'appellation " **mise à jour** " est synonyme d'entrée/sortie (ou " in out ") : elle signifie que la procédure qui déclare un tel paramètre a besoin de connaître sa valeur pour pouvoir s'exécuter

correctement, et modifiera cette valeur au cours de son exécution, de sorte que la variable effective utilisée dans l'appel aura changé de valeur par rapport à celle qu'elle avait initialement. On parle aussi parfois de " donnée modifiée ".

Un **paramètre de sortie** est intégralement affecté au sein de la procédure, indépendamment de toute valeur initiale : utiliser une variable en tant que paramètre effectif de sortie d'une procédure équivaut à une initialisation de cette variable par une affectation.

Dans l'un comme dans l'autre cas (sortie ou mise à jour), **le paramètre doit être passé " par référence " ou encore " par alias "** : à cette fin, on peut utiliser le mot-clé `ByRef` en lieu et place du mot-clé `ByVal` employé pour les paramètres d'entrée ; cependant, le mot-clé `ByRef` étant la valeur par défaut lorsqu'on ne précise pas le mode de passage, on ne l'utilise quasiment jamais explicitement.

```
1  Sub Permuter(a As Integer, b As Integer)
2  'ROLE : permuter les valeurs des deux valeurs entières spécifiées
3  'M.A J. a, b
4      Dim tmp As Integer 'variable auxiliaire utilisée pour faire la permutation
5      tmp = a
6      a = b
7      b = tmp
8  End Sub
9
10 Sub TesterPermuter()
11 'ROLE : tester la fonction Max
12     Dim i As Integer, j As Integer
13     Afficher "Saisir␣un␣premier␣nombre␣:␣"
14     Saisir i, 2
15     Afficher "Saisir␣un␣second␣nombre␣:␣"
16     Saisir j, 5
17     Afficher "Avant␣permutation,␣i␣=␣" & i & ",␣j␣=␣" & j
18     Permuter i, j
19     Afficher "Après␣permutation,␣i␣=␣" & i & ",␣j␣=␣" & j
20 End Sub
```

Lorsqu'un paramètre est passé par référence, le paramètre formel se comporte comme un alias du paramètre effectif, c'est-à-dire une 2^{me} nom désignant la même variable. Techniquement, le paramètre formel est utilisé comme variable locale initialisée à chaque appel, non avec la valeur de la variable utilisée comme paramètre effectif (ce qui était le cas du passage par valeur), mais avec **l'adresse interne de l'emplacement-mémoire** de cette variable. Ainsi, toute modification de valeur du paramètre formel dans la procédure est en fait réalisée sur le paramètre effectif lui-même.

En dessous du rôle de la procédure, on spécifie obligatoirement les modes de passage des paramètres par un ou plusieurs commentaires commençant par `SORTIE` ou par `M A J` (ou `M. A J.`). L'usage de l'abréviation `M A J` (avec espaces ou points séparateurs) permet d'utiliser un " mot-clé " de 6 caractères comme pour les autres modes de passage `ENTREE` et `SORTIE`. De même, le fait de mettre les deux-points derrière `ROLE` permet d'aligner verticalement toutes les précisions indiquées derrière ces mots, que l'on retrouvera pour toutes les déclarations de procédures.
Attention : Un paramètre effectif de sortie ou de mise à jour doit obligatoirement être une variable, à laquelle sera affecté le résultat du calcul effectué dans la procédure.

5 Paramètres de type Tableau

En Visual Basic, les tableaux sont obligatoirement passés par référence. L'emploi du mot-clé ByVal provoque donc une erreur de syntaxe.

Pour signifier qu'un tableau est passé en paramètre, on utilise une paire de parenthèses vide. Voici l'exemple de deux procédures et d'une fonction permettant respectivement de générer un tableau d'entiers contenant des valeurs aléatoires entre 1 et 100, d'affecter en bloc 2 tableaux, et de comparer en bloc 2 tableaux. Rappelons que l'affectation en bloc d'un tableau dans un autre, et la comparaison en bloc de 2 tableaux n'existent pas en Visual BASIC.

```
1  Sub GenererTableau(Tableau() As Double, ByVal Taille As Integer)
2  'ROLE : génère un tableau contenant des valeurs aléatoires entières
3  '       entre 1 et 100 inclus
4  'SORTIE Tableau
5  'ENTREE Taille : nombre d'éléments du tableau
6      Dim i As Integer
7      For i = 1 To Taille: Tableau(i) = Int(Rnd * 100) + 1: Next i
8  End Sub
9
10 Sub AffecterTableau(TableauCible() As Double, _
11                     TableauSource() As Double, ByVal Taille As Integer)
12 'ROLE : Affecte à un tableau-cible le contenu d'un tableau-source
13 'ENTREE Taille : nombre d'éléments de chaque tableau
14 '       TableauSource
15 'SORTIE TableauCible
16     Dim i As Integer
17     For i = 1 To Taille: TableauCible(i) = TableauSource(i): Next i
18 End Sub
19
20 Function TableauxEgaux(Tableau1() As Double, Tableau2() As Double, _
21                        ByVal Taille As Integer) As Boolean
22 'ROLE : renvoie VRAI si les 2 tableaux spécifiés sont égaux
23 'ENTREE Taille : nombre d'éléments de chaque tableau
24 '       Tableau1, Tableau2 : les 2 tableaux à comparer
25     Dim i As Integer, Egaux As Boolean
26     i = 1: Egaux = True 'égalité a priori ; on boucle jusqu'à ce qu'on infirme
27     Do While (i <= Taille) And Egaux
28     'Tester si les i-èmes éléments des tableaux sont égaux
29         If Tableau1(i) = Tableau2(i) Then i = i + 1 Else Egaux = False
30     Loop
31     TableauxEgaux = Egaux
32 End Function
33
34 Sub Testertableaux()
35 'ROLE : tester les procédures et fonctions précédentes
36
37     Const T = 4
38     Dim T1(1 To T) As Double, T2(1 To T) As Double
39
40     EffacerEcran "Tests␣sur␣les␣tableaux"
41
```

```
42     Afficher "1.Test␣procédure␣GenererTableau"
43      GenererTableau T1, T
44      Afficher "tableau␣T1␣:"
45      Afficher T1
46
47     Afficher "2.Test␣fonction␣TableauxEgaux"
48      GenererTableau T2, T
49      Afficher "tableau␣T2␣:"
50      Afficher T2
51      If TableauxEgaux(T1, T2, T) Then Afficher "T1␣=␣T2" Else Afficher "T1␣<>␣T2"
52
53     Afficher "3.Test␣fonction␣AffecterTableau"
54      AffecterTableau T2, T1, T
55      Afficher "Après␣T2<-T1,␣T2␣vaut␣:"
56      Afficher T2
57      If TableauxEgaux(T1, T2, T) Then Afficher "T1␣=␣T2" Else Afficher "T1␣<>␣T2"
58 End Sub
```

La procédure de test suivante illustre un exemple d'appel des procédures et fonction précédentes.

```
1  Sub Testertableaux()
2  'ROLE : tester les procédures et fonctions précédentes
3
4      Const T = 4
5      Dim T1(1 To T) As Double, T2(1 To T) As Double
6
7      EffacerEcran "Tests␣sur␣les␣tableaux"
8
9     Afficher "1.Test␣procédure␣GenererTableau"
10     GenererTableau T1, T
11     Afficher "tableau␣T1␣:"
12     Afficher T1
13
14    Afficher "2.Test␣fonction␣TableauxEgaux"
15     GenererTableau T2, T
16     Afficher "tableau␣T2␣:"
17     Afficher T2
18     If TableauxEgaux(T1, T2, T) Then Afficher "T1␣=␣T2" Else Afficher "T1␣<>␣T2"
19
20    Afficher "3.Test␣fonction␣AffecterTableau"
21     AffecterTableau T2, T1, T
22     Afficher "Après␣T2<-T1,␣T2␣vaut␣:"
23     Afficher T2
24     If TableauxEgaux(T1, T2, T) Then Afficher "T1␣=␣T2" Else Afficher "T1␣<>␣T2"
25 End Sub
```

6 Paramètres de type Enregistrement

Les enregistrements peuvent être passés comme paramètres d'entrée, sortie ou mise à jour de toute procédure ou fonction. Cependant, comme une variable de type enregistrement a généralement vocation à gérer de nombreux attributs et donc à occuper un espace-mémoire susceptible

d'être important, **on a l'habitude de toujours passer par référence de tels paramètres, même s'ils sont en entrée**.

Par ailleurs, le type de retour d'une fonction peut tout à fait être de type enregistrement.

7 Constantes et variables globales

En Visual BASIC, les variables globales susceptibles d'être utilisées par des procédures ou fonctions d'un module doivent être déclarées en début de module, avant la première définition de procédure ou fonction.

En général, on limite les déclarations globales à des constantes, ce qui permet de facilement rassembler en tête de module les constantes symboliques susceptibles d'être facilement modifiées lors de versions successives du programme.

8 Portée des définitions (Public, Private)

La portée des définitions des constantes, variables, types enregistrements, procédures et fonctions définit les zones du projet Visual BASIC dans lesquels celles-ci seront visibles, et donc utilisables.

8.1 Constantes et variables (Private par défaut)

Par défaut, les constantes et variables définies en début d'un module ne sont visibles que ce modules. On peut étendre cette visibilité à tous les modules constituant le projet en précédant le mot-clé **`Const`** ou **`Dim`** par le mot-clé **`Public`**.

Attention : Il est très fortement déconseillé de déclarer publique une variable globale car cela constitue un trou de sécurité important : en effet, en cas d'erreur sur le contenu d'une telle variable à un moment donné, il est plus difficile de détecter l'instruction en cause puisqu'elle peut figurer dans n'importe quel module du projet, et non pas simplement dans le module qui la déclare.

8.2 Types enregistrements (toujours Public)

Lorsqu'un type enregistrement est défini dans un module, il est visible depuis tous les modules du projet et permet donc de déclarer des variables de ce type.

8.3 Procédures et fonctions (Public par défaut)

Par défaut, les procédures et fonctions définies dans un module sont visibles dans tous les modules constituant le projet. On peut restreindre cette visibilité au seul module dans lequel elles sont définies en précédant le mot-clé **`Sub`** ou **`Function`** par le mot-clé **`Private`**. En particulier, une macro (procédure sans paramètre) définie comme privée ne pourra pas être associée à un

bouton de feuille de calcul. Il est recommandé d'utiliser **`Private`** pour qualifier des procédures ou fonctions dont on est certain qu'elles ne présentent aucun intérêt à être appelées depuis d'autres modules du projet.

9 Autres particularités du Visual BASIC

Le Visual BASIC possède quelques particularités que nous n'approfondissons pas ici mais qui peuvent apporter des fonctionnalités intéressantes dans certains cas particuliers :

- définition de *paramètres d'entrée facultatifs* dont les paramètres formels reçoivent une valeur par défaut en cas d'absence du paramètre effectif correspondant dans l'instruction d'appel (cf. aide sur les mots-clés `Optional` et `ParamArray`, et sur la fonction prédéfinie **`IsMissing`**)
-
- définition de *variables locales rémanentes* capables de conserver leur valeur entre deux appels successifs à une même procédure (cf. aide sur le mot-clé **`Static`**) ; ce type de variable permet notamment de compter combien de fois une procédure a été exécutée.
-

10 Débogage de procédures ou fonctions (Pas à pas principal/détaillé)

L'exécution d'un appel de procédure en mode "pas à pas" peut se faire de 2 manières :

- pour rentrer dans la procédure (et donc s'arrêter sur la ligne d'en-tête définissant la procédure) : valider l'option **Pas à pas détaillé** du menu Débogage, ou cliquer sur le bouton Pas à pas détaillé (barre Débogage), ou frapper la touche **F8**
-
- pour exécuter en bloc l'ensemble de la procédure (et donc s'arrêter sur l'instruction qui la suit dans le bloc d'instructions courant) : valider l'option **Pas à pas principal** du menu Débogage, ou cliquer sur le bouton Pas à pas principal (barre Débogage), ou frapper la touche Maj-F8
-

On peut également poursuivre l'exécution de la procédure en cours et s'arrêter sur l'instruction suivant son instruction d'appel dans la procédure appelante en validant l'option **Pas à pas sortant** du menu Débogage, ou en cliquant sur le bouton Pas à pas sortant (barre Débogage), ou en frappant la touche **Ctrl-Maj-F8**.