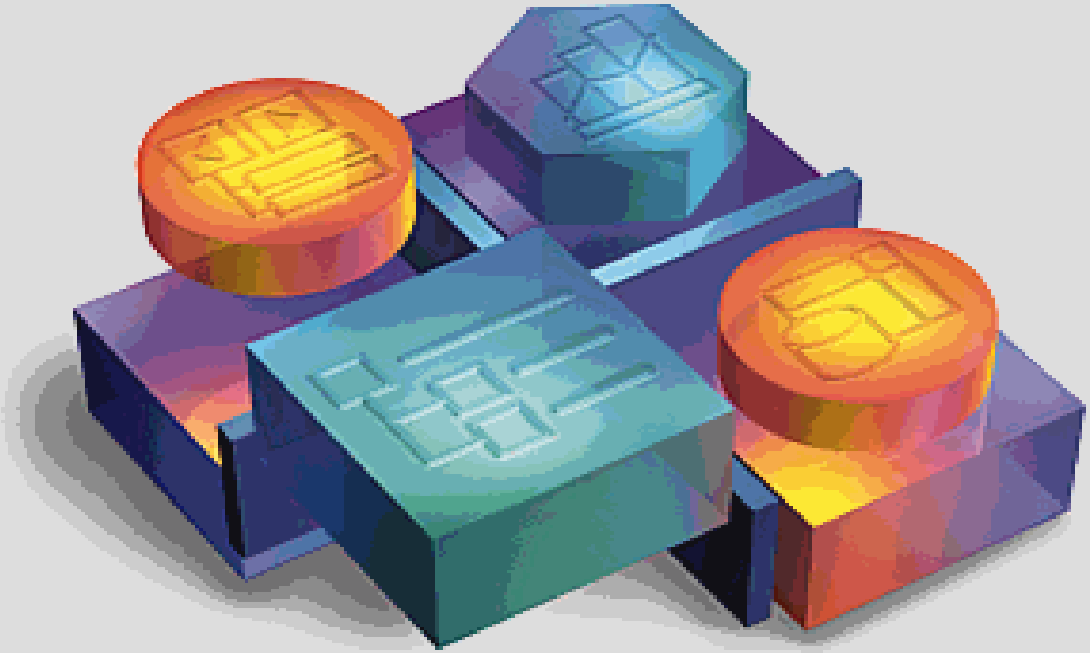
Le langage Basic



Variables, types et constantes

Déclaration des variables

En développement, on entend par variable une donnée définie dans un contexte donné ayant un type défini.

Autrement dit, une variable est la représentation d'une valeur au sens large du terme.

On appelle déclaration le fait de définir la variable avant de l'utiliser, dimensionnement : lui donner un type.

En Visual Basic la déclaration des variables n'est pas obligatoire tant que l'option Explicit n'est pas précisé, par défaut le type de ces variables est « Variant ».

L'instruction de déclaration est :

Nom de la variable As Type de la variable

La Portée des variables

La notion de portée, parfois appelée visibilité, défini les limites d'accessibilité d'une variable. Il existe plusieurs instructions de déclaration selon la portée désirée et la déclaration ne se fait pas au même endroit.

Instruction	Déclaration	Commentaires
Private	Module	Visible par tout le code du module mais inaccessible depuis un autre module
Public	Module (standard)	Visible par tout le code du projet. Ne se déclare que dans les modules standard.
Dim	Fonction	Uniquement dans la fonction ou elle est déclarée. Si utilisée au niveau module, équivaut à Private
Static	Fonction	Uniquement dans la fonction ou elle est déclarée. N'est pas détruite à la fin de la fonction

Les types Numériques

Type	Etendue	Taille en octets
Byte (octet)	0 à 255	1
Integer (entier)	-32768 à 32767	2
Long (entier long)	-2 147 483 648 à 2 147 483 647	4
Single (réel simple à virgule flottante)	-3,402823E38 à 1,401298E-45 et 1,401298E-45 à 3,402823E38	4
Double (réel double à virgule flottante)	-1,7976931348623 1E308 à 4,940656458412 47E-324 et 4,940656458412 47E-324 à 1,7976931348623 1E308	8
Decimal	+/-79 228 162 514 264 337 593 543 950 335	12

Opérateurs arithmétiques

Opérateur	Commentaire
+	Addition de deux nombres
-	Soustraction de deux nombres
*	Multiplication de deux nombres
/	Division de deux nombres, le dénominateur ne peut être nul.
\	Division entière. Renvoie la partie entière du résultat
^	Élévation à la puissance du nombre de gauche par celui de droite
Mod	Renvoie le reste de la division du nombre de gauche par celui de droite.

Opérateurs de comparaison

Les opérateurs de comparaison sont tout aussi connus que les opérateurs arithmétiques

Opérateur	vrai si	faux si
< (inférieur à)	*expression1 < expression2*	*expression1 >= expression2*
<= (inférieur ou égal à)	*expression1 <= expression2*	*expression1 > expression2*
> (supérieur à)	*expression1 > expression2*	*expression1 <= expression2*
>= (supérieur ou égal à)	*expression1 >= expression2*	*expression1 < expression2*
= (égal à)	*expression1 = expression2*	*expression1 <> expression2*
<> (différent de)	*expression1 <> expression2*	*expression1 = expression2*

Les **opérateurs logiques**

En Visual Basic, les mêmes opérateur logique servent pour la logique et la logique binaire. Ce n'est pas forcément une bonne chose en terme de lisibilité mais avec l'habitude on évite les erreurs que cela pourrait entraîner.

Les opérateurs logiques s'utilisent toujours sous la forme :
résultat = *expression1* **opérateur** *expression2*

A l'exception de l'opérateur Not. Expression1 et expression2 doivent donc renvoyer des valeurs booléenne (vrai/faux) et le résultat en sera une aussi

Opérateur Not

Applique la négation logique, c'est-à-dire inverse le résultat d'une expression. De la forme :
résultat = **Not** *expression*

Opérateur And

Vérifie que les deux expressions sont vraies.

Expression1	Expression2	Résultat
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Si une expression renvoie Null, le résultat sera Null si l'autre expression est vraie, faux sinon.

Opérateur Or

Vérifie qu'au moins une des deux expressions sont vraies.

Expression1	Expression2	Résultat
Vrai	Vr	Vrai
Vra	Faux	Vrai
Fau	Vrai	Vrai
Faux		Faux

Si une expression renvoie Null, le résultat sera Null si l'autre expression est fausse, vrai sinon.

Opérateur XOr

Vérifie l'exclusion logique c'est-à-dire que les deux expressions renvoient un résultat différent.

Expression1	Expression2	Résultat
Vrai	Vrai	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Si une expression renvoie Null, le résultat sera Null.

Opérateur Imp

Vérifie l'implication logique c'est-à-dire que si l'expression1 est vraie, l'expression2 doit l'être aussi.

Expression1	Expression2	Résultat
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Vrai
Faux	Faux	Vrai
Null	Vrai	Vrai
Null	Faux	Null
Vrai	Null	Null
Faux	Null	Faux
Null	Null	Null

Les chaînes de caractères

Le type est String. Il existe deux types de chaînes :

- les chaînes de longueur variable peuvent contenir environ 2 milliards de caractères.
- les chaîne de longueur fixe peuvent contenir de 1 à environ 64 Ko de caractères.

Exemple

```
'Chaîne de longueur variable
Dim Ville As String
'Chaine de longueur fixe (20 caractères)
Dim Nom As String * 20
```

L'opérateur de concaténation est le signe &

Like, permet de comparer deux chaînes de caractères

"Bonjour" like "Au revoir" => False

Autres types

Booléen

Le type est Boolean. La variable peut prendre la valeur True (Vrai) ou False (Faux) qui est sa valeur par défaut. Elle occupe deux octets.

Currency

Le type est Currency. La variable peut prendre les valeurs de -922 337 203 685 477,5808 à 922 337 203 685 477,5807. Ce type permet d'exprimer des valeurs en monnaies. Elle occupe huit octets.

Autres types

Date

Le type est Date. La variable peut prendre les valeurs de date et d'heure du 01/01/1970 au 31/12/9999. Elle occupe huit octets.

Variant

Les variables de type Variant peuvent contenir des données de toutes sortes ainsi que les valeurs Empty, Error, Nothing et Null. Utiliser le type de donnée Variant offre plus de souplesse dans le traitement des données. Par exemple, si une variable de type variant contient des chiffres, il peut s'agir de leur valeur réelle ou de leur représentation sous forme de chaîne, selon le contexte.

Le type Date et heure

Vous pouvez assigner une date à une variable date comme une chaîne de caractère :

```
Dim MyDate As Date
MyDate = "12/1/2012"
```

Le format de la chaîne peut différer en fonction du pays paramétré sur le système d'exploitation.

Pour éviter cela il est possible d'utiliser la fonction DateSerial & TimeSerial :

```
MyTime = TimeSerial (12, 18, 2)
```

Détails d'extraction de dates et heures

Les fonctions suivantes forme l'ensemble des fonctions `DateSerial et TimeSerial`:

Day(MyDate)
Month(MyDate)
Year(MyDate)
Weekday(MyDate)
Hour(MyTime)
Minute(MyTime)
Second(MyTime)

Type Objet

Le type est object. Pour créer une variable destinée à contenir un objet, commencez par déclarer la variable comme étant type Objet puis affectez-lui un objet.

•Pour déclarer une variable Objet : si le type de l'objet est inconnu, utilisez la syntaxe :
InstructionDéclaration NomVariable As Object
si le type de l'objet est connu, utilisez la syntaxe :
InstructionDéclaration NomVariable As TypeObjet

Exemple

```
Sub Variables_Objet()

'Objet est déclaré en tant qu'objet
'NomClient est déclaré en tant que feuille de calcul
'Graph est déclaré en tant que graphique
Dim Objet As Object
Dim NomClient As Worksheet
Dim Graph As Chart

End Sub
```

•Pour affecter un objet à une variable Objet, utilisez l'instruction Set :
Set NomVariable = ObjetàAffecter

Exemple : Déclaration d'une variable ZoneDeTest destinée à contenir un objet Range puis affectation des cellules A1 à B10 à cette variables :

```
Dim ZoneDeTest As Range
Set ZoneDeTest = Range("A1 : B10")
```

•Pour mettre fin à l'association entre une variable et un objet précis, utilisez la syntaxe suivante :
Set NomVariable = Nothing

L’option « Option Explicit »

Il est possible d’imposer la déclaration des variables avant utilisation dans la section de déclaration de chaque module.

On utilise pour cela la déclaration de l'entête « option explicit », par défaut si l’on n’utilise pas cette déclaration, toute variable non déclaré sera du type variant.

Cela oblige à utiliser les mots clés Dim, Public, Private, si le type de la variable n’est pas précisé, alors le type par défaut est variant.

Types composites

Il est possible de définir des types composites, appelés types utilisateurs à l'aide de l'instruction Type… End Type.

L'intérêt est évidemment de manipuler plusieurs variables connexes à l'aide d'une seule variable. Ce type étant ensuite considéré comme n'importe quel type, vous pouvez déclarez des variables de ce type, des tableaux, le renvoyer dans des fonctions, etc…

La définition d'un type utilisateur se fait obligatoirement au niveau du module. Dans un module standard, il peut être public ou privé, dans un module objet il ne peut être que privé.

La déclaration se fait sous la forme :

Portee Type *NomType*
Element As Type
Element As Type
….
End Type

Les éléments qui composent le type (appelés membres) peuvent être de n'importe quels types prédéfinis, des tableaux ou d'autres types utilisateurs. Il est donc possible d'obtenir des structures extrêmement complexes.

N'oubliez pas que cette définition ne suffit pas pour manipuler la structure, vous devez déclarer des variables de ce type pour l'utiliser effectivement.

Types composites

```
Public Type Fichier
    Nom As String
    Repertoire As String
    DateCration As Date
    Taille As Long
End Type

Public Sub test()

Dim FichiersExcel() As Fichier, compteur As Long, fTemp As String

fTemp = Dir("d:\svg\jmarc\*.xls", vbNormal)
Do Until fTemp = ""
    ReDim Preserve FichiersExcel(0 To compteur)
    FichiersExcel(compteur).Nom = fTemp
    FichiersExcel(compteur).Repertoire = "d:\svg\jmarc\*.xls"
    FichiersExcel(compteur).DateCration = FileDateTime("d:\svg\jmarc\" &
fTemp)
    FichiersExcel(compteur).Taille = FileLen("d:\svg\jmarc\" & fTemp)
    compteur = compteur + 1
    fTemp = Dir
Loop
MsgBox FichiersExcel(0).Nom & vbNewLine & FichiersExcel(0).Taille

End Sub
```

Type énumération

Une énumération est un groupement de constantes entières connexes. Elle est toujours déclarée au niveau du module. L'intérêt repose surtout sur la lisibilité du code, on peut indifféremment utiliser la valeur numérique ou le membre de l'énumération. Elle se déclare comme suit :

Portée Enum *Name*

NomMembre = [*ConstanteEntiere*]

NomMembre = [*ConstanteEntiere*]

....

End Enum

Type énumération

Les valeurs d'une énumération peuvent être utilisées soit directement, soit à l'aide de variables, passées comme argument ou renvoyées par une fonction. Si les valeurs des constantes ne sont pas précisées dans la déclaration, le premier membre vaut zéro, chaque membre suivant étant égal à la valeur du membre précédent agrémenté de 1.

VBA utilise de nombreuses énumérations intégrées.

```
Public Enum Mois
    Janvier = 1
    Février = 2
    Mars = 3
    Avril = 4
    Mai = 5
    Juin = 6
    Juillet = 7
    Aout = 8
    Septembre = 9
    Octobre = 10
    Novembre = 11
    Décembre = 12
End Enum

Public Enum Facteur
    kilo = 1024
    mega = 1048576
    giga = 1073741824
End Enum
```

Pour créer une variable, vous devez la déclarer, c'est à dire lui affecter un nom. Vous pouvez ensuite utiliser ce nom pour modifier la valeur de la variable, ou encore utilisez cette variable pour des calculs.

En VBA il existe deux types de déclaration :

- implicite
- explicite

Déclarations implicites

Elles se font directement par l'affectation d'une valeur à un nom de variable. Le type de données est alors le type par défaut, soit **Variant**

Exemple

```
X = 12
Prix = 1200
Prénom = "Fabrice"
```

Déclarations explicites

Elles nécessitent l'utilisation d'une **instruction de déclaration** (Dim, Public, Private, global...). Si le type de la variable n'est pas précisé, le type par défaut, soit Variant, est alors affecté à la variable.

Il est possible d'imposer la déclaration implicite des variables en utilisant l'instruction **Option Explicit** dans la section de déclaration de chaque module.

Exemple

```
Dim X
Private Prix As Double
Public Prénom As String
```

Pour optimiser la rapidité du code Basic, il est préférable de déclarer les variables de façon explicite.

Déclarations explicites du type

Le type de la variable est précisé, lors de la déclaration de celle-ci, après le mot clé As.

Il est possible de déclarer plusieurs variables dans une même instruction, mais attention, le type de donnée ne sera pris en compte que pour la dernière variable.

Exemple

```
Dim Nom, Prénom, Adresse As String
Dim Ville, région As String
```

Dans l'exemple ci-dessus, seule les variables Adresse et région seront de type String, alors que les autres seront de type Variant.

Déclarations implicites du type

Le type de variable se trouve déclaré par l'emploi d'un **suffixe** au moment de son utilisation ou par l'instruction **DefType**.

- **Emploi d'un suffixe**

Vous devez ajouter l'un des caractères suivants au nom de la variable :

Suffixe	Type de données
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Exemple

Déclaration d'une variable étant de type Chaîne (String)

```
Dim Adresse$
```

Déclaration d'une variable étant de type Monnaie (Currency)

```
Dim Argent@
```

DefType

Ces instructions s'utilisent dans la zone de déclaration du module pour définir les types de données par défaut des variables dont les noms commencent par les caractères spécifiés.

Liste des instructions DefType :

Instruction	Type de données
DefBool	Boolean
DefDbl	Double
DefInt	Integer
DefDate	Date
DefLng	Long
DefStr	String
DefCur	Currency
DefObj	Object
DefSng	Single
DefVar	Variant
DefByte	Byte

Exemple : Toutes les variables qui commencent par une lettre comprise entre A et D seront des variables de type Chaîne (String)

DefStr A-D

Les tableaux

Vous pouvez créer une variable tableau lorsque vous avez besoin de travailler avec un groupe de valeurs.
Pour créer une variable tableau, utilisez la syntaxe suivante :

InstructionDeDéclaration NomDuTableau (indices)

Où pour (indices):

- Si vous oubliez cet argument : tableau à une dimension
- Si vous indiquez un chiffre ; tableau avec le nombre d'élément avec un nombre d'élément précis et des numéros d'indices spécifiques

Par défaut, le plus petit indice d'un tableau est **0, ceci peut etre modifié par "option base 0 ou option base 1"**.

Les tableaux

Il est possible de créer des tableaux multidimensionnel, le principe reste le même que pour un tableau unidimensionnel.

Pour cela on va déclarer les différentes dimensions séparés par des virgules.

Dim montab(dim1,dim2,dim3,...,dimN)

Dim montab (5,5) as Integer

Les tableaux

Il est possible de redimensionner un tableau en cour d'execution de notre macro, ceci peut être réalisé avec la fonction redim

Redim [Preserve] montab(Dim1) As Type

Le mot clé « Preserve », permet de conserver les valeurs déjà inscrites dans le tableau, sans ce mot clé spécifique, le tableau est remis à Zéro.

Ex : dim tableau(10) as Integer
redim preserve tableau(14) as Integer

Les fonctions LBound() et UBound() retournent respectivement l'index le plus bas et l'index le plus haut du tableau.

```
Dim MyArray(10) As Integer
' ... des instructions
Dim n As Integer
n = 47
Redim MyArray(n) As Integer
MsgBox(LBound(MyArray))  ' affiche : 0
MsgBox(UBound(MyArray))  ' affiche : 47
```

Pour un tableau multi-dimention, on donnera en paramètre des fonctions le numéro de la dimension

```
Dim MyArray(10, 13 to 28) As Integer
MsgBox(LBound(MyArray, 2))  ' affiche : 13
MsgBox(UBound(MyArray, 2))  ' affiche : 28
```

Instruction Erase

L 'instruction Erase, permet d'effacer un tableau.

Dans le cas des tableaux fixes, les valeurs sont réinitialisés mais le tableau garde les dimensions qui sont définies.

Dans le cas des tableaux dynamiques, le tableau est réinitialisé complètement (perte des valeurs et des dimensions).

Syntaxe : **Erase nomtableau**

Les constantes

[Public] Const NomConstante [As Type] = Valeur

Exemple :

```
Const PI as Double = 3.14159
```

Les Structures de contrôles
Et les boucles

L'instruction Do...Loop

Exécute un bloc d'instructions un nombre de fois indéterminé

Syntaxe 1

Les instructions sont exécutées aussi longtemps que la condition renvoie True.

```
Do
    Instructions
Loop
```

Syntaxe 2

Les instructions sont exécutées une première fois sans condition puis aussi longtemps que la condition renvoie True.

Do
    Instructions
Loop While Condition

```
Dim Nombre

Do

    Nombre = InputBox("Entrer un nombre suppérieur à 100")

Loop While (Not IsNumeric(Nombre) Or Nombre <= 100)
```

Exemple : Le code suivant demande à l'utilisateur de saisir un nombre tant que celui-ci n'est pas numérique ou n'est pas suppérieur à 100.

Syntaxe 3

Les instructions sont exécutées jusqu'à ce que la condition renvoie True (aussi longtemps que la condition renvoie False).

```
Do Until Condition
    Instructions
Loop
```

Syntaxe 4

Les instructions sont exécutées une première fois sans condition puis jusqu'à ce que la condition renvoie True

```
Do
    Instructions
Loop Until Condition
```

Exemple : Le code suivant demande à l'utilisateur de saisir un nombre tant que celui-ci soit numérique et suppérieur à 100.

L'instruction While…Wend

Exécute une série d'instructions dans une boucle tant que la condition spécifiée est vérifiée.

Syntaxe

While Condition
    Instructions
Wend

Exemple :

```
'On demande un prix tant qu'il n'est pas
'renseigné correctement

While IsEmpty(Range("A1")) Or Not IsNumeric(Range("A1"))

    Range("A1") = InputBox("Saisir le prix")

Wend
```

L'instruction For…Next

Exécute un bloc d'instructions en fonction de la valeur d'un compteur.

Syntaxe

For compteur=départ **To** fin **step incrément**
    Instructions
Next

Exemple : Affichage des 5 éléments d'un tableau

```
Dim tableau(5) As String

tableau(0) = "Lille"
tableau(1) = "Paris"
tableau(2) = "Marseille"
tableau(3) = "Nantes"
tableau(4) = "Bordeaux"

For i = 0 To 4 Step 1

    MsgBox tableau(i)

Next
```

L'instruction For Each…Next

Exécute un bloc d'instructions pour chaque élément d'une collection d'objet ou d'un tableau.

Syntaxe

For Each Elément In Groupe
**   Instructions**
Next Elément

Exemple : Affiche "OK" dans toutes les cellules d'une plage

```
Dim cellule As Range

For Each cellule In Range("A1:B10")

    cellule = "OK"

Next
```

Quitter les structures de contrôle

L'instruction **Exit For** permet de quitter directement une boucle For ou For Each tandis que **Exit Do** quitte directement une boucle Do.

Exemple : On quitter la boucle quand la cellule n'est plus vide

```
Dim cellule As Range
Dim i As Integer

i = 0

'Cette boucle serait infinie sans Exit Do,
'Car la valeur de i est toujours 0 !
Do While i < 1

    If Range("A1") Is Not Empty Then
        Exit Do
    End If

    Range("A1") = InputBox("Tappez un nombre")

Loop
```

If…Then

L'instruction IF

Permet d'exécuter des instructions en fonction du résultat d'une condition

- **If...Then**

If condition **Then** instruction

S'il y a plusieurs instructions, séparez-les par le signe de ponctuation : (deux-points). Cette syntaxe est surtout utilisée pour des tests courts et simples.

Exemple : Si la cellule A1 est vide, alors envoi d'un bip sonore et d'un message

```
Sub Test_Cellule_A1()

    If IsEmpty(Range("A1")) Then Beep: MsgBox "Celulle vide"

End Sub
```

If…Then…End If

- **If...Then...End If**

If condition **Then**
    Instruction 1
    Instruction 2
End If

Exemple

```
Sub Test_Cellule_A1()

    'Si la cellule A1 est non vide
    'alors elle est mise en gras et coloriée en rouge
    If Not IsEmpty(Range("A1")) Then
        Range.Font.Bold = True
        Range.Interior.ColorIndex = 3
    End If

End Sub
```

If…Then…Else…End If

If...Then...Else...End If

If condition **Then**

    Instruction

Else

    Instruction

End If

Exemple : Si la cellule A1 est inférieur à 10 alors on affiche Bravo sinon Perdu.

```
Sub Test_Cellule_A1()

    'Si la cellule A1 est suppérieur à 10
    'alors afficher Bravo sinon Perdu
    If Not Range("A1").Value > 10 Then
        MsgBox "Bravo"
    Else
        MsgBox "Perdu"
    End If

End Sub
```

If…Then…ElseIf…Else…End If

If...Then...ElseIf...Else...End If

If condition **Then**
    Instruction
ElseIf condition **then**
    Instruction
ElseIf condition **then**
    Instruction
Else
    Instruction
End If

```
Sub Test_Cellule_A1()

    'Si la cellule A1 est suppérieur à 100
    'alors afficher Bravo ! Si la cellule A1
    'est comprise entre 10 et 100 on affiche
    'Moyen ! Si la cellule A1 est inférieur à
    '10 on affiche Perdu ! Autrement Erreur !
    If Range("A1").Value > 100 Then
        MsgBox "Bravo !"
    Else
        If Range("A1") < 10 Then
            MsgBox "Perdu !"
        ElseIf Range("A1") >= 10 Then
            MsgBox "Moyen !"
        Else
            MsgBox "Erreur !"
        End If
    End If

End Sub
```

Exemple : Si la cellule A1 est suppérieur à 100 alors on affiche Bravo !, si la cellule A1 est inférieur à 10 on affiche Perdu !, si la cellule A1 est compris entre 10 et 100 on affiche Moyen ! Autrement on affiche Erreur !

L'instruction Select Case

Exécute une des séquences d'instructions spécifiées en fonction de la valeur d'une expression

Select Case ExpressionTestée
    Case ListeExpression
        Instruction
    Case ListeExpression
        Instruction
    Case Else
        Instruction
End Select

```
Function Total(Quantités, HT, Port)

    Select Case Quantités
        Case 1
            Total = HT + Port
        Case 2 To 3
            Total = HT * Quantités
        Case 4 To 5
            Total = HT * Quantités * 0.95
        Case Is = 6
            Total = HT * Quantités * 0.9
        Case Else
            Total = "Erreur dans la quantité"
    End Select

End Function
```

Liste Expression peut prendre l'une des formes suivantes :

- valeur (ex : Case 10)
- liste des valeurs (ex : Case 1 To 5)
- expression conditionnelle (ex : Case Is >= 5)

Exemple : Création d'une fonction pour calculer un total avec ou sans remise, en fonction des quantités commandées :

Les structures compactes

reponse=IIF(test,si test=vrai, si test=faux)

L'instruction iif, est une forme contracté du « if then else », ceci est une fonction qui récupère un résultat en fonction de l'évaluation booléenne du test.

Exemple :
reponse=iif(n>10,"gagné","perdu")

Les structures compactes

Choose, renvoie une valeur d'une liste de choix en fonction de l'index passé en premier argument, cette instruction remplace avantageusement une instruction case simple.

CHOOSE (Index,Choix1,Choix2,...., Choixn)

EqvColor=Choose(index, "noir","blanc","rouge","vert","bleu")

Les structures compactes

Switch, cette fonction renvoie une valeur ou une expression fonction de l'expression qui lui est associé dans une liste d'expressions.

SWITCH(expr1,valeur1,expr2,valeur2,.....,exprn,valeurn)

signe=switch(valeur<0,-1,valeur=0,0,valeur>0,1)

With…End With

L'instruction **With…End With** est l'instruction de bloc Visual Basic. L'instruction de bloc permet de faire référence à un objet une seule fois pour tout le bloc de ligne de codes qu'il contient. Ceci permet une augmentation de vitesse du code puisque la référence n'est résolue qu'une fois et une meilleure lisibilité.

```
Dim Valeur As Variant

With Range("B2")
    .Font.ColorIndex = 7
    .Interior.ColorIndex = 5
    Valeur = .Value
    .BorderAround ColorIndex:=3, Weight:=xlThick
    .FormulaLocal = "=Somme(A1:A10)"
End With
```

Procédures et fonctions

Les procédures « Sub »

Il existe deux types de procédures Sub

- les procédures Sub générales
- les procédures sub événementielles
-

Une procédure générale est une procédure déclarée dans un module (généralement un module standard). L'appel d'une telle procédure est défini explicitement dans le code.

Une procédure événementielle est une procédure associée à un événement d'un objet.

Syntaxe d'une procédure Sub

```
[Private | Public | Friend] [Static] Sub NomProc
([liste d'arguments])
    séquence d'instructions
End Sub
```

Les Fonctions

Les procédures Function, plus couramment appelés fonctions, renvoient une valeur, telle que le résultat d'un calcul. La valeur retournée doit porter le nom de la fonction.
Le langage Basic comporte de multiples fonctions intégrés telles que les fonctions se rapportant aux dates (day, week, year,...)

En plus de ces fonctions intégrées, vous pouvez créer vos propres fonctions personnalisées.

Syntaxe d'une procédure Function

```
[Private | Public | Friend] [Static] Function NomProc
([liste d'arguments]) [As Type]
    séquence d'instructions
End Function
```

Arguments des procédures

Les arguments sont utilisés pour transmettre aux procédures des paramètres sous formes de données. Le nombre d'argument peut varier de 0 à plusieurs.

Pour déclarer un argument, il suffit de spécifier son nom. Néanmoins la syntaxe de déclaration d'un argument est la suivante

[Optional] [ByVal | Byref] [ParamArray]
variable [As type]

L'option Byval indique que l'argument est passé par valeur. La procédure accède à une copie de la variable, la valeur initiale de celle-ci n'est donc pas modifiée par la procédure à laquelle elle est passée.

L'option Byref indique que l'argument est passé par référence. La procédure peut ainsi accéder à la variable proprement dite, la valeur réelle de celle-ci peut, de ce fait, être modifiée par la procédure à laquelle elle a été passée.

Variable : précise le nom de l'argument. Pour les variables tableau, ne pas préciser les dimensions.

Type : précise le type de données de l'argument passé à la procédure (**boolean, integer, long,...**).

L'option Optionnal indique que l'argument est facultatif. Les arguments facultatifs doivent être de type Variant .

Ce modificateur précise que l'argument est optionnel, c'est-à-dire qu'il peut être transmis comme il peut ne pas l'être. Les règles suivantes s'appliquent :

- Lorsqu'un argument est marqué comme optionnel, tout les arguments qui le suivent dans la déclaration doivent être marqués comme étant optionnels.
- Les arguments optionnels ne peuvent pas être utilisés en même temps que les tableaux d'arguments (ParamArray)
- L'absence du paramètre peut être testé avec la fonction IsMissing si son type n'est pas un type valeur.
- Les arguments optionnels peuvent avoir une valeur par défaut. Dans ce cas IsMissing renvoie toujours False.

Les deux syntaxes qui suivent sont équivalentes :

```
Private Function Arrondir(ByVal Valeur As Single, Optional NombreDeDecimale
As Variant) As Single

    If IsMissing(NombreDeDecimale) Then NombreDeDecimale = 1
    Arrondir = Valeur * (10 ^ NombreDeDecimale)
    Arrondir = Int(Arrondir + 0.49) * (10 ^ -NombreDeDecimale)

End Function
```

Notez que pour que cette fonction fonctionne correctement, l'argument NombreDeDecimale est de type Variant et non de type Integer.

Le mot clé **ParamArray** : est utilisé comme dernier argument de la liste pour indiquer que celui-ci est un tableau facultatif d'élément de type **Variant** . Il ne peut être utilisé avec les mots clés ByVal, Byref ou Optional.

Indique que l'argument est un tableau variant de paramètre. Ce mot clé ne peut être utilisé que sur le dernier argument. Ce mot clé n'est guère utilisé car il présente quelques difficultés de manipulation, car il représente en fait un nombre illimité d'arguments. Prenons l'exemple suivant :

```
Private Function Minimum(ParamArray Elements() As Variant) As Variant

Dim cmpt As Long
If UBound(Elements) > -1 Then 'au moins un argument
    For cmpt = LBound(Elements) To UBound(Elements)
        If IsNumeric(Elements(cmpt)) Then
            If Not IsEmpty(Minimum) Then
                If Minimum > Elements(cmpt) Then Minimum = Elements(cmpt)
            Else
                Minimum = Elements(cmpt)
            End If
        End If
    Next cmpt
End If

End Function

Public Sub Test()

    MsgBox Minimum("toto", -1, True, 124, Range("B2"), 25.471)

End Sub
```

Comme nous le voyons, si la fonction déclare bien un argument de type tableau, l'appel utilise lui une liste d'éléments. Le tableau étant de type variant, les éléments peuvent être de n'importe quel type.

Notez que l'indice du premier argument est toujours 0.

Appel d'une procédure

Syntaxe

[CALL] NomProc [liste d'arguments]

Si le mot clé **Call** est indiqué, vous devez placer la liste d'arguments entre parenthèses.

- Pour stocker le résultat d'une fonction dans une variable, utilisez la syntaxe suivante
  variable = NomFunc ([liste d'arguments])
- Pour appeler une procédure d'un autre module, utiliser la syntaxe suivante
  NomDuModule.NomDeLaProcédure

A connaître ...

. Les commentaires

Les commentaires permettent de documenter les codes VBA afin de les rendre plus lisibles.

REM commentaire
ou
' commentaire

Dés validation de la ligne de commentaire, celle-ci s'affiche par défaut en **vert**.

. Le caractère de continuation

Un instruction VBA peut être ecrite sur plusieurs ligne en utilisant un trait de soulignement " **_** " précédé d'un espace.

Exemple :

```
'Demande de saisie d'un prix tant que celui-ci
'n'est pas renseigné ou est incorrect

Do While IsEmpty(Prix) Or Not IsNumeric(Prix) _
    Or Prix < 50

    Prix = InputBox("Saisir le Prix")

Loop
```

Les retraits

Les retraits (ou tabulations) permettent une meilleur lisibilité du code. Il notamment important de les utiliser dans les structures de contrôle (surtout si il y a des imbriquations) et les structures de décisions.

- Pour générer des retraits, utiliser la touche **Tabulation**.
- Pour revenir à la tabulation précédente, uilisez la touche **Shift** + **Tabulation**.
- Pour modifier la taille de la tabulation (quatre espaces par défaut) :
  - Selectionnez **Options** à partir du menu **Outils**
  - Cliquez sur l'onglet **Editeur** et modifiez la zone **Largeur de la tabulation**

Les noms de procédures, variables et constantes

Les noms des procédures, des constantes, des variables et des arguments doivent respecter les règles suivantes :

- Le premier caractère doit être une lettre
- Les minuscules et majuscules ne sont pas différenciées (les lettres accentuées sont acceptées) bien que la casse soit respectée
- Ne pas utiliser de noms réservés à Visual Basic, comme Dim ou Integer
- Ne pas employer de point, d'espace, de !, de $, de # et d'@
- Un nom ne peut pas compter plus de 255 caractères
- Ne pas indiquer plusieurs fois les mêmes noms de varables et de constantes dans un même niveau de portée

Les fonctions VBA

Fonctions mathématiques

Fonction	Signification	exemple
`+,-,*,/`	+, -, $\frac{3}{2}$, /	
x `\` y	x div y	`5 \ 2` $\frac{23}{11}$ 2
x `Mod` y	x mod y	`5 mod 2` $\frac{23}{11}$ 1
x `^` y	x^y	`2^3` $\frac{23}{11}$ 8
`Abs(`x`)`	$\|x\|$	`Abs(-3.2)` $\frac{23}{11}$ 3.2
`Cos(`x`)`	cos x	`Cos(0)` $\frac{23}{11}$ 1
`Exp(`x`)`	e^x	`Exp(1)` $\frac{23}{11}$ 2.718282
`Fix(`x`)`	retourne l'entier sans les décimales	`Fix(-3.2)` $\frac{23}{11}$ -3
`Int(`x`)`	E(x) partie entière	`Int(-3.2)` $\frac{23}{11}$ -4
`Log(`x`)`	ln x	`Log(Exp(1))` $\frac{23}{11}$ 1
`Rnd()`	retourne un nombre aléatoire entre 0 et 1	`1 + Rnd() * 5`
`Sin(`x`)`	sin x	`Sin(0)` $\frac{23}{11}$ 0

Chaînes de caractères

Fonction	Signification	exemple
Instr(*s*, *ss***)**	position de la première occurrence de la sous-chaîne *ss* dans la chaîne *s*	**Instr("ok ok","ok")** 1
Lcase(*s***)**	convertit en minuscules tous les caractères alphabétiques de *s*	**Lcase("OK !")** "ok !"
Left(*s*,*n***)**	chaîne correspondant aux *n* caractères de gauche de *s*	**Left("Hello",4)** "Hell"
Len(*s***)**	nombre de caractères de la chaîne *s*	**Len("bonjour !")** 9
LTrim(*n***)**	chaîne où les espaces qui sont au début de *s* sont supprimés	**LTrim(" ok ! ")** "ok ! "
Mid(*s*, n_1, n_2**)**	chaîne correspondant aux n_2 caractères pris à partir de la position n_1 dans *s*	**Mid("Salut",3,2)** "lu"
Right(*s*,*n***)**	chaîne correspondant aux *n* caractères de droite de *s*	**Right("Hello",2)** "lo"
RTrim(*s***)**	chaîne où les espaces qui sont à la fin de *s* sont supprimés	**RTrim(" ok ! ")** " ok !"
String(*n*,*c***)**	chaîne composée de *n* fois le caractère *c*	**String(3,"a")** "aaa"
Trim(*s***)**	chaîne où les espaces qui sont au début et à la fin de *s* sont supprimés	**Trim(" ok ! ")** "ok !"
Ucase(*s***)**	convertit en majuscules tous les caractères alphabétiques de *s*	**Ucase("hé")** "HÉ"

Caractères, formatage, conversion

Fonction	Signification	Exemple
Asc(*s***)**	code numérique (ASCII) du 1e caractère de la chaîne *s*	**Asc("BONJOUR")** 66 **Asc("0")** 48
Chr(*n***)**	caractère correspondant au code *n*	**Chr(48)** "0" **Chr(13)** " " (retour à la ligne)
C...(*expr***)**	convertit *expr* dans le type spécifié par ...	**CStr(8/3)** "2.66667 " **CInt(8/3)** 3 **CLng(8/3)** 2.666666667 **CDbl(8/3)** 2.666666667
IsNumeric(*s***)**	Retourne true si *s* correspond à la représentation d'un nombre	**IsNumeric("-3,2")** true **IsNumeric("alpha")** false
Format(*expr***,** *format***)**	chaîne correspondant au formatage de *expr* selon *format*	**Format(18,"#,###.00 €")** "18,00 €" **Format(5100,"00 000")** "05 100"

Financières

Fonction	Signification
DDB	Amortissement d'un bien pour une année
FV	Valeur future d'une annuité basée sur des paiements constants et périodiques et à un taux d'intérêt constant
IPmt	Montant des intérêts pour une période donnée d'une annuité basée sur des paiements constants et périodiques et un taux d'intérêt constant
IRR	Taux de retour interne pour une série de mouvements de trésorerie périodiques
NPer	Nombre de périodes pour une annuité basée sur des paiements constants, périodiques et soumis à un taux d'intérêt constant
NPV	Valeur nette actuelle d'un investissement, calculée en fonction d'une série de mouvements de trésorerie périodiques
Pmt	Montant d'une annuité basé sur des paiements constants, périodiques et soumis à un taux d'intérêt constant
PV	Valeur actuelle d'une annuité basée sur des paiements futurs constants et périodiques et à un taux d'intérêt constant
Rate	Taux d'intérêt par période pour une annuité
SLN	Amortissement direct d'un bien sur une période donnée

Dates

Fonction	Signification	Exemple
`Date()`	Date courante	`If Date() > DateValue("1/1/00") Then …`
`DateAdd(`*type*`,`*n*`,`*date*`)`	Ajoute une durée de *type* à *date*	`DateAdd("d",2,"1/2/01")` 23/11 3/2/01 `DateAdd("m",2,"1/2/01")` 23/11 1/4/01
`DateDiff(`*type*`,`d_1`,`d_2`)`	Intervalle de temps de *type* entre d_1 et d_2	`DateDiff("d","1/1/01","1/2/01")` 23/11 31 `DateDiff("m","1/1/01","1/2/01")` 23/11 1
`DateValue(`*expr*`)`	Entier correspondant à la date *expr*	`DateValue("1/1/00")+1` 23/11 `2/1/00`
`Format(`*date*`,`*format*`)`	Formatte *date* selon *format*	`Format("1/1/02","dddd d mmm yy")` 23/11 mardi 1 janv 02 `Format(0,"dd/mm/yyyy")` 23/11 30/12/1899 `Format("1/6/02", "mmmm")` 23/11 `juin`
`IsDate(`*expr*`)`	Détermine si *expr* correspond à une date	`IsDate("1/12/01")` 23/11 true `IsDate("1/14/01")` 23/11 false `IsDate("32/1/01")` 23/11 true `IsDate("demain")` 23/11 false
`Now()`	Date et heure courantes	

Heures

Fonction	Signification	Exemple
dateDiff(*type*,d_1,d_2**)**	Intervalle de temps de *type* entre d_1 et d_2	**dateDiff("h","1/1/01","1/2/01")** 744 (= 31 x 24)
Time()	Heure courante	**If Time() > TimeValue("12:00:00") Then** …
TimeValue(*expr***)**	Nombre réel correspondant à l'heure *expr*	**Cdbl(TimeValue("12:00:00"))** 0.5 **TimeValue("12:00:00")+1/(24*60)** **12:01:00**
Format(*heure*,*format***)**	Formatte *heure* selon *format*	**Format("12:05:00","hh \h mm")** 12 h 05
Hour(*heure***)**	Nombre d'heures dans *heure*	**Hour("12:05:00")** 12
Minute(*heure***)**	Nombre de minutes dans *heure*	**Minute("12:05:00")** 5
Second(*heure***)**	Nombre de secondes dans *heure*	**Second("12:05:00")** 0

Système

Fonction ou instruction	Signification	Exemple
Shell(*commande***)**	lance une commande MS-DOS et retourne 1 si la commande s'est correctement exécutée	**e = Shell("C:\WINDOWS\CALC.EXE")** **If e = 1 Then** …
Dir(*chemin***)**	Teste l'existence du répertoire ou du fichier indiqué par le chemin *chemin*	**Dir("c:\autoexec.bat")** 23 11 **"c:\autoexec.bat"** **Dir("c:\windoze.daube")** 23 11 **""**
FileCopy $chemin_1$, $chemin_2$	Copie le fichier indiqué par dans chemin $chemin_1$ dans $chemin_2$	**FileCopy "c:\temp\f.txt" "c:\temp\f.bak"**
MkDir *chemin*	Crée le répertoire indiqué par le chemin *chemin*	**MkDir "c:\temp\bak"**
RmDir *chemin*	Supprime le répertoire indiqué par le chemin *chemin*	**RmDir "c:\temp\bak"**

Fonctions sur les fichiers

- 2 types de fichiers :
  - texte (lisibles par un éditeur de texte)
  - binaires
- 2 modes d'accès aux données dans les fichiers :
  - accès séquentiel : pour accéder à la donnée *i*, il faut déjà avoir accédé à la donnée *i* – 1
  - accès direct (ou aléatoire) : on peut accéder directement à une donnée dans un fichier si l'on connaît sa position dans le fichier

Fonctions sur les fichiers

Fonction ou instruction	Signification	Exemple
Open *chemin* **For** **As #***num*	Ouvre le fichier indiqué par *chemin* en lecture, en écriture ou ajout sur le canal n ◦*num*	**Open "c:\data.txt" For Input As #1**
Input #*num* var_1,..., var_n	Lit les variables var_1,..., var_n dans le fichier ouvert sur le canal *num*	**Input #1, nom, prenom**
Line Input #*num var*	Lit une ligne dans le fichier ouvert sur le canal *num* et la stocke dans la variable *var*	**Input #1, ligne**
Write #*num* $expr_1$,..., $expr_n$	Ecrit les expr. $expr_1$,..., $expr_n$ dans le fichier ouvert sur le canal *num*	**Write #1, nom, prenom**
Close #*num*	Ferme le fichier ouvert sur le canal *num*	**Close #1**
EOF(*num***)**	Retourne true si l'on est à la fin du fichier ouvert sur le canal *num*	**If EOF(1) Then** …