3 LE LANGAGE

3.1 Syntaxe de base

Le retour chariot est le séparateur d'instruction. Les instructions sont impérativement contenues dans des procédures ou des fonctions. Seules les déclarations (variables, constantes, structures, énumérations…) peuvent être effectuées en dehors.

- '_' : permet de continuer une instruction sur la ligne suivante lorsque l'on souhaite afficher le code sur une largeur d'écran ;
- ':' : permet de mettre plusieurs instructions sur une même ligne. Cela équivaut au retour chariot. Cette syntaxe n'est pas recommandée et très peu utilisée ;
- ''' : permet d'insérer des commentaires. Chaque ligne commentée doit être précédée de la cote.

Il faut garder à l'esprit qu'un code bien écrit est plus facile à maintenir car plus lisible. En outre, la concision de la syntaxe n'est pas proportionnelle aux performances d'exécution.

3.2 Les variables

La valeur contenue dans une variable peut être modifiée. Le choix du nom est libre dès lors qu'il ne contient pas d'espace et qu'il ne s'agit pas d'un mot réservé. Le nom doit commencer impérativement par une lettre et comporter un maximum de 255 caractères.

3.2.1 Les types

VB contient de très nombreux types de variables si l'on prend en compte les types objets. On trouve les types de base tel que :

- Integer ;
- String ;
- Single, Double ;
- Long ;
- Byte ;
- Boolean ;
- Date…

On utilise la syntaxe suivante pour déclarer une variable. Nous verrons plus loin qu'il existe des variantes pour modifier la portée :

```
Dim [NomVar] As Type
```

Ex :

```
Dim intVal As Integer
```

On peut déclarer des variables de type objet. On utilise la même syntaxe que pour les types de base, mais on invoque une classe.

```
Dim MonForm As Form
Dim MaBase As DataBase
```

La déclaration d'objet est un eu plus subtile qu'il n'y paraît sans être particulièrement complexe. Une variable objet est un « lien » vers une zone mémoire. Lors de la déclaration, VB crée la variable sans allouer immédiatement l'espace mémoire. Cette phase intervient ultérieurement. Il est possible de réserver l'espace mémoire dès la déclaration en utilisant le mot réservé *New*.

```
Dim MaBase As New Form
```

On trouve également des types génériques tels que :

- Variant ;
- Object.

Le type *Variant* peut accueillir tous les types élémentaires : entier, chaîne, long, date ainsi que les valeurs spéciales *Empty* et *Null*. Il n'est pas défini tant qu'on n'y affecte pas de valeur. Il prend le plus petit type possible pour accueillir la valeur affectée. Ainsi, si on affecte 38954 à un variant, il prendra le type *Long* (entier long) car le type *Integer* est trop petit. On dit alors qu'un Variant a pris le sous-type Long.
Toutefois, ce type est déconseillé car on ne maîtrise pas son contenu et il est difficile de prévoir tous les comportements.

Le type « Object » est aux objets ce que « Variant » est aux types de base. Il peut accueillir toute référence à un objet. En effet, les variables objets sont des **références** (liens) vers les objets.

La fonction *VarType* renvoie le sous-type d'un Variant sous forme de constante VB (*vbInteger*, *vbBoolean*,…). Consulter *Dim* dans l'aide pour plus d'informations sur les types et leur taille en mémoire.

Attention : il est possible de déclarer plusieurs variables à l'aide d'une seule instruction *Dim*. Toutefois, il est impératif de préciser le type pour chaque variable, sans quoi le type sera *Variant* par défaut.

```
Dim strNom, strPrenom, strCommentaire As string
```

Seule *strCommentaire* est de type chaîne de caractère car le type est précisé. Les autres variables sont de type *Variant*.

3.2.2 Déclaration implicite

VB autorise de ne pas déclarer des variables. Lorsque vous utilisez un nom qui n'a jamais été utilisé auparavant, VB considère qu'il s'agit d'une variable et la déclare automatiquement. Cette méthode est très vivement déconseillée car elle est génératrice d'erreurs souvent difficile à tracer.

Les variables utilisées sans déclaration sont de type *variant*. On peut attribuer un type par défaut à l'aide de *DefInt*, *DefBool*, *DefDbl*… placer en tête de module.

```
DefInt I – N, Y
```

Les variables ayant un nom commençant par I jusqu'à N et Y seront de type *Integer*.

Dans le cas d'une déclaration implicite, les variables ont une portée Procédure.

3.2.3 Déclaration explicite

Cette méthode est à privilégier. Utilisez le menu Outils / Option, onglet Editeur et cochez la case « Déclaration des variables obligatoire ». La modification sera active au prochain lancement de VB. Vous pouvez aussi écrire *Option Explicit* en tête de tous les modules. Cela signifie que la déclaration est obligatoire. L'activation de l'option précise à VB d'ajouter la mention citée automatiquement.
La déclaration des variables est désormais impérative sous peine d'erreur de compilation.

On utilise les mots réservés suivant pour déclarer des variables :

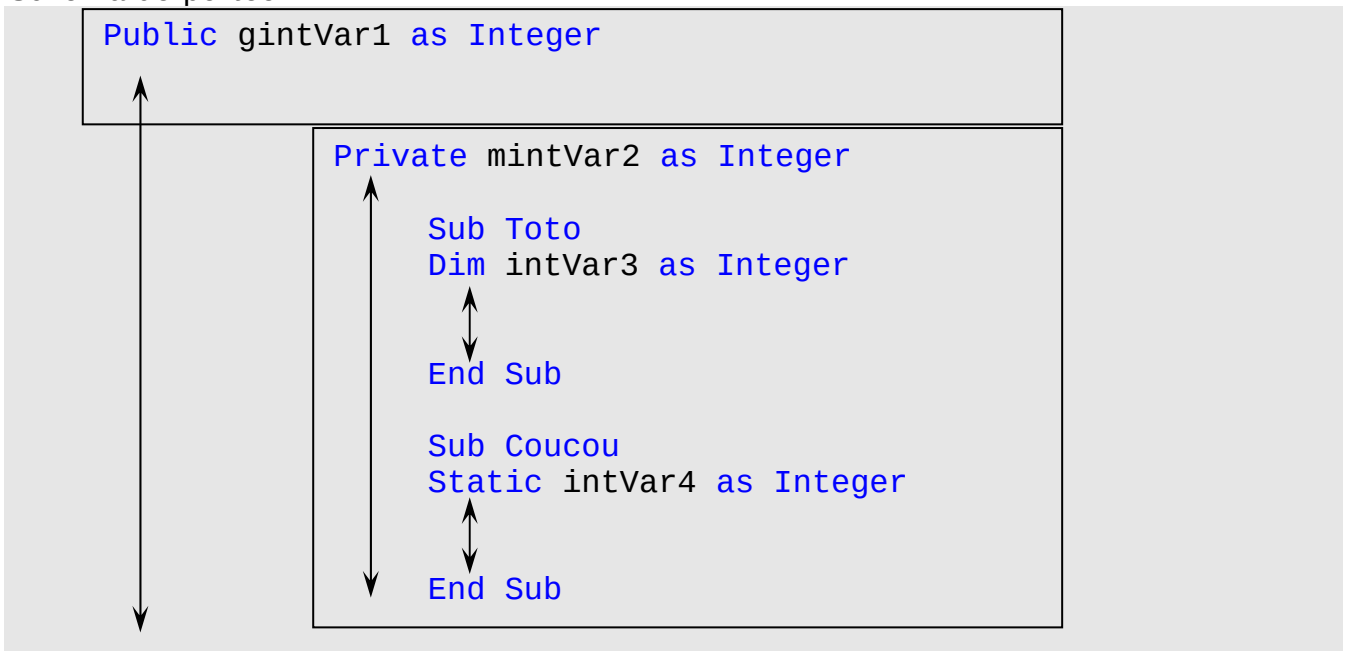
- Dim, Private ;
- Public ;
- Static (uniquement au niveau procédure.)

La portée des variables dépend du mot réservé utilisé et de l'endroit où est déclarée la variable.

On définit trois portées :

Globale ou publique : la variable est visible dans tout le projet, i.e. tous les modules et formulaires. On déclare ces variables au début des modules standards à l'aide du mot réservé *Public* ;
Module : la variable est visible dans le module où elle est déclarée, ie dans toutes les procédures et fonctions du module. Elle est déclarée en tête de module à l'aide des mots réservés *Private* ou *Dim* ;
Procédure ou fonction : la variable n'est visible que dans la procédure ou la fonction où elle est déclarée. On utilise *Dim* ou *Static*.

Schéma de portée :



Les variables sont initialisées automatiquement : 0 en numérique, «» en chaîne, et *nothing* en objet.

Une variable n'existe que dans sa portée. Ainsi, une variable procédure est créée dans la procédure avec *Dim* ou *Private*, et détruite lorsque l'on sort de la procédure. Elle n'est évidemment plus visible car elle n'existe plus.

Les variables déclarées avec *Static* répondent aux règles de portée précisées ci-dessus, i.e. qu'elles ne sont pas visibles en dehors de leur portée. Toutefois, leur valeur est conservée après l'exécution de la procédure jusqu'au prochain appel. Etudiez l'exemple suivant (on utilise une seule procédure coucou à la fois) :

```
Sub Toto() 'Cette procédure appelle Coucou
Dim i as Integer
        For i = 1 to 3
                Coucou          'Selon la déclaration des variables de
        Next i                  'Coucou, les résultats changent.
End Sub

Sub Coucou() 'Premier cas
Dim intNum as Integer
        intNum = intNum + 1
        MsgBox intNum'toujours = 1
End Sub

Sub Coucou() 'Deuxième cas
Static intNum as Integer
        intNum = intNum + 1
        MsgBox intNum'= 1, = 2 etc
End Sub
```

Dans le deuxième cas, *intNum* conserve la valeur de l'appel précédent : 1 au premier appel, 2 au deuxième (1 du précédent plus 1 de l'appel courant)…

Les Tableaux :

Les tableaux sont des variables particulières car elles contiennent plusieurs éléments sous un même nom. Les différents éléments sont accessibles par un index. Un tableau peut avoir jusqu'à 60 dimensions.
Les éléments contenus dans un tableau sont impérativement du même type. On peut déclarer des tableaux de *Variant*. Dans ce cas, on peut stocker un sous-type différent dans chaque case du tableau. Encore une fois, il convient d'être prudent dans l'utilisation des *Variant*.

Une dimension :

```
Dim sngMontant(1 To 12) as Single 'Une dimension d'index 1 à 12
Dim intMois as Integer

intMois = 5
sngMontant(intMois) = 9500.25 'Affectation d'une valeur dans la case d'index 5 du tableau
```

Par défaut, l'index de départ d'un tableau est 0. Ainsi, un tableau déclaré comme suit :

```
Dim strNom(5) as String
```

est un tableau à une dimension d'index allant de 0 à 5 (6 cases.)

Plusieurs dimensions :

```
Dim sngMontant(1 To 12, 1990 To 2000) as Single
'Première dimension allant des index 1 à 12, seconde dimension allant de 1990 à 2000
```

```
Dim strNom(5,9) as String 'Première dimension allant des index 0 à 5 (6 cases), seconde dimension allant de 0 à 9 (10 cases)
```

Les tableaux déclarés jusqu'ici sont de taille fixe, i.e. que le nombre de dimension et le nombre de case par dimension ne peuvent plus être modifiés. Il convient alors d'être prudent dans la déclaration pour ne pas déclarer de tableaux trop petits.

VB offre la possibilité d'utiliser des tableaux dynamiques. Il sera possible de modifier la taille et le nombre de dimension après la déclaration.

Sans dimension : tableaux dynamiques

Ces tableaux sont déclarés avec des parenthèses vides. Le compilateur comprend qu'il s'agit d'un tableau.

```
Dim sngMontant() as Single
```

Un tableau ne peut pas être utilisé sans taille. VB met à disposition un mot réservé qui permet de donner une taille à un tableau dynamique.

```
ReDim  Permet de redimensionner un tableau, mais perte des valeurs.
```

La ligne suivante donne deux dimensions à notre tableau :

```
ReDim sngMontant(5,9)
```

Notez que le type est déjà déclaré. On peut utiliser ce mot réservé autant que nécessaire au cours d'une procédure, mais le contenu des cases est réinitialisé à chaque utilisation du mot ReDim. Il faut lui adjoindre le mot réservé *Preserve* pour conserver le contenu.

```
ReDim Preserve sngMontant(5,15) 'pour conserver les valeurs.
```

Ce système permet d'étendre la taille d'un tableau au cours de l'exécution. Il est évident que si le programmeur diminue la taille du tableau, le contenu des cases supprimées est perdu.

Remarque : Seule la dernière dimension peut être étendue sans perte de données. Cette contrainte est liée au stockage en mémoire d'un tableau qui se fait de manière linéaire. Ainsi, la première case de la deuxième dimension suit la dernière case de la première.

On détermine les bornes (index min et max) des dimensions d'un tableau à l'aide des mots réservés *Ubound* et *Lbound* qui permettent de connaître respectivement les bornes supérieures et inférieures d'un tableau.

```
Ubound(Nom du tableau[, Dimension])
```

Ex :

```
Ubound(sngMontant, 2) ‘Renvoie 15 selon le ReDim précédent
```

Exemple :

```
Dim sngMontant() as Single
ReDim sngMontant(1 To 2)
SngMontant(1) = 9600
SngMontant(2) = 7420
ReDim Preserve sngMontant(1 To Ubound(sngMontant) + 1)
SngMontant(Ubound(sngMontant)) = 14300
```

La fonction Array renvoie une variable de type Variant contenant un tableau :

```
Dim A as Variant
Dim B as Integer

A = Array(3, 5, 14)
B = A(2)          ‘B = 14
```

Les structures personnalisées :

VB autorise la création de types de variables personnalisées. Ces structures sont composées de membres de type élémentaires. On utilise le mot *Type* pour déclarer une structure. Un type ne peut être déclaré qu'au niveau module.

```
Public Type Navire
        Nom As String * 32
        Type As String
        TirantEau As Single
End Type
```

On accède aux membres d'une structure selon la syntaxe objet (point '.').

```
Sub Toto()
Dim V1 as Navire
Dim V2 as Navire
        V1.Nom = "Granvillaise"
        V1.Type = "Bisquine"
        V1.TirantEau = 2.75

        V2 = V1
        MsgBox V2. Type                ‘Affiche Bisquine
End Sub
```

3.3 Les constantes et énumérations

3.3.1 Constantes

Les constantes contiennent des valeurs définies lors de la conception et n'évoluent plus. Elles répondent aux mêmes règles de portée que les variables. Elles ne peuvent être que privées dans les modules de classes.
Elles ont pour but de rendre le code plus lisible en remplaçant une valeur par un nom plus parlant.

Public const STR_CHAMP_NOM = 2 permet d'identifier un champ de table par un nom explicite plutôt que par un index. Le compilateur remplacera STR_CHAMP_NOM par 2 partout où il le rencontrera dans le code.

Lors de la compilation, les références aux constantes utilisées dans le code sont remplacées par la valeur de chaque constante. Ainsi, cette méthode ne pénalise pas la rapidité d'exécution du programme compilé.

On peut utiliser d'autres types : &H pour héxa, &0 pour octal et #3/5/2000# pour les dates.

3.3.2 Enumérations

Les énumérations sont des structures de constantes entières. Par défaut, leurs membres sont numérotés à partir de 0 et incrémentés de 1 en 1. Elles ne peuvent apparaître qu'au niveau module.

```
Public Enum Palette
        BLEU            '= 0
        BLANC '= 1
        VERT            '= 2
End Enum
```

Ou encore

```
Public Enum Palette
        BLEU            '= 0
        BLANC = 3       '= 3
        VERT            '= 2
End Enum
```

On peut ensuite les utiliser comme un type de données. Par exemple :

```
Sub Toto(monParam as Palette)
        [Code…]
End Sub
```

3.4 Convention de nomenclature

Bien qu'il n'y ait pas d'obligation syntaxique, **certaines préconisations de nomenclature permettent de clarifier le code. Ainsi, des préfixes renseignent sur le type et la portée des variables.**
On utilise fréquemment *g* pour une variable globale (publique), *m* pour une variable de portée module et rien pour une variable de portée procédure.

On ajoute également trois lettres pour informer sur le type de la variable :
int, *lng*, *str*, *frm*, *rst*, *dbl* etc. On obtient ainsi :

```
        gintAnnee 'variable en integer de portée globale (ou publique)
        strNom 'variable chaîne de portée procédure
```

```
mdblRatio 'variable booléenne de portée module
rstProduit etc.
```

Les constantes sont déclarées en majuscule et peuvent être préfixées :

```
CST_IDX_CODE
SQL_SEL_PRODUIT
SQL_DEL_OUTIL
```

3.5 Fonctions et procédures

3.5.1 Les procédures

La procédure est l'unité d'exécution de traitement. Elle ne retourne aucune valeur après avoir exécuté les instructions qu'elle contient.

Le déclenchement des procédures est évènementielle (cf. I 3°) ou par appel. Elles sont déclenchées par les actions de l'utilisateur sur l'interface lorsqu'elles sont événementielles. Aussi, une procédure peut en appeler d'autre afin de structurer le code.

Par exemple, on écrira une procédure qui sauvegarde des données dans un fichier ini ou la base de registre, puis on l'appellera à partir de plusieurs autres procédures. Cela évite de réécrire plusieurs fois le même code.

Les procédures peuvent avoir deux portées : globale (publique), visible dans l'ensemble du projet, ou privée, visible uniquement dans le module où elles se trouvent. Elles sont déclarées comme suit :

```
Public Sub Toto() est une procédure globale ;
Private Sub Titi() est une procédure privée (module);
Public Function Coucou() est une fonction globale ;
Private Function Coucou() est une fonction privée (module).
```

Si les mots réservés Public ou Private sont omis, la procédure est publique par défaut. Une procédure est un bloc d'instruction qui a un début et une fin.

```
Public Sub Save_Param()
        [Déclaration des variables locales…]

        [instructions…]

End Sub
```

Il ne peut jamais y avoir d'instruction exécutable en dehors d'une procédure.

3.5.2 Les paramètres

Un paramètre est une sorte de variable exposée par une procédure. Lors de l'appel de la procédure, on donne une valeur au paramètre, valeur qui sera manipulée par la procédure.

Ces paramètres peuvent avoir tous les types des variables, et être passés par référence ou par valeur, être impératifs ou optionnels.

```
Public Sub Toto(ByRef p_strNom As String, ByVal p_intAge As Integer)
```

Ces deux paramètres sont impératifs. Par référence, la valeur du paramètre peut être modifiée par la procédure, par valeur, on passe une copie de la valeur, celle-ci n'est donc pas modifiée lorsque la procédure appelante reprend l'exécution. Le passage par défaut est par référence si omis.

Exemple :

```
Sub Toto()

        Save_Param « Admin » 'Appel de la procédure et initialization du paramètre

End Sub

Public Sub Save_Param(p_strValeur as String)

        MsgBox p_strValeur 'Affiche Admin

End Sub
```

Un paramètre optionnel est passé comme suit : *Optional p_strNom As String*. Il peut être omis lors de l'appel de la procédure. **Une fonction de VB permet de savoir si un paramètre optionnel a été passé, toutefois, cette fonction ne fonctionne que si le paramètre est de type Variant.** En effet, les autres types sont automatiquement initialisés, la fonction ne peut alors savoir si le paramètre de type chaîne par exemple est omis ou réellement une chaîne vide :

```
IsMissing(Param).
```

On peut donner une valeur par défaut à un paramètre optionnel :

```
        Optional p_strAge As Integer = 12
```

Si le paramètre n'est pas passé lors de l'appel, la procédure utilisera la valeur par défaut (12 dans cet exemple).

Le mot réservé *ParamArray* permet de passer une liste de valeurs sans que l'on connaisse sa taille à l'avance. Il ne peut être utilisé que sur le dernier paramètre :

```
        Sub Toto(p_intVal as Interger, ParamArray p_strListe())
```

P_strListe() est manipulé comme un tableau au sein de la procédure. On peut en déterminer les bornes avec *Lbound* et *Ubound*. Il n'a qu'une seule dimension et est obligatoirement de type Variant.

3.5.3 Les fonctions

Les fonctions se caractérisent par le fait qu'elles retournent une valeur à la procédure ou à la fonction appelante.

Les fonctions suivent les mêmes règles de déclaration et de portée que les procédures. Le type retourné est précisé dans la définition de la fonction. S'il est omis, le type est variant.

```
Public Function Toto(p_intAge As Integer) As Single
```

Une des nouveautés de VB6 est de permettre aux fonctions de retourner un tableau.

```
Public Function Toto(p_intAge As Integer) As Single()
```

Pour définir la valeur retournée par la fonction, on affecte la valeur au nom de la fonction. En effet, le nom de la fonction peut être utilisé comme une variable locale du type retourné par la fonction.

Dans l'exemple précédent, la ligne *Toto = 1.438* placée dans la fonction fera que cette dernière renverra 1.438 à l'appelant.

Voici un exemple de passage de paramètres *ByVal* ou *ByRef* (par défaut) :

```
Private Sub Form_DblClick()
Dim i as Interger

        i = 2
        Carre i
        MsgBox i                ‘Affiche 4 ByRef et 2 ByVal

End Sub
```

Le code suivant est une procédure et non une fonction. Il ne retourne donc pas de valeur. Pourtant, on voit bien que le passage de paramètre par référence modifie la valeur de la variable de la procédure appelante.

```
Sub Carre(pintVal as Long)      ou (ByVal pintVal as Long)
        PintVal = pintVal * pintVal
End Sub
```

Remarque : Lorsque VB fait un calcul, il utilise le type le plus précis des opérandes. Si un opérande est un Integer et un autre est un Long, VB va faire son calcul dans un Long. Il est important de voir si les résultats estimés rentrent dans le type le plus précis des opérandes. Exemple :

```
IntOpe1 = 200 ‘Integer
IntOpe2 = 200 ‘Integer

LngResult = intOpe1 * intOpe2           ‘Erreur car 200 * 200 ne rentre pas dans un integer bien que LngResult soit de type long
```

Utilisation d'une fonction puis d'un paramètre optionnel.
On passe à la fonction qui suit un tableau de single. On boucle sur ce tableau pour en sommer les éléments. Notez que le type retourné par la fonction est un single ; il serait prudent de recourir à un double qui est un type plus grand. Cette précaution éviterait d'éventuels dépassements de capacité.

```
Function Somme(ParamArray N()) As Single
```

```
Dim i As Integer
        For i = 0 to Ubound(N)
                Somme = Somme + N(i)
        Next i
End Function
```

```
Private Sub Form_BblClick()
        MsgBox Somme()
        MsgBox Somme(1)
        MsgBox Somme(1,2,3)
End Sub
```

3.6 Structures de contrôles

On appelle structures de contrôles les blocs d'instructions élémentaires tels que les tests ou les boucles. Tous les langages possèdent ces structures, ce n'est pas une spécificité de VB. Seule la syntaxe varie d'un langage à l'autre.

3.6.1 Les structures conditionnelles

↳ *If…Then :*

Il s'agit de la structure de test la plus fréquente. La *condition1* est une expression qui renvoie vrai ou faux. Selon l'évaluation de cette expression, le compilateur exécute un des blocs *Instructions*.

La syntaxe complète de cette structure est la suivante :

```
If condition1 Then
        Instructions
        …
ElseIf condition2 Then   '"Sinon si" est optionnel
        Instructions
        …
Else                                    '"Sinon " est optionnel
        Instructions
        …
End If
```

Lors de l'exécution, si une des conditions est remplie, les autres sont ignorées. Par exemple, si la première condition est vérifiée, le premier bloc est exécuté puis on sort de la structure de test (End If). Les blocs suivant ne sont pas exécutés.

Remarque : L'ensemble des expressions *condition* sont évaluée. Elles doivent donc être correctes.

Exemple :

```
Dim intNote As Integer
intNote = 13

If intNote < 10 Then
        MsgBox "Recalé"
ElseIf intNote >= 10 And intNote < 12 Then
```

```
        MsgBox “Passable”
ElseIf intNote >= 12 And intNote < 14 Then
        MsgBox “Bien”
Else                    ‘Tous les autres cas soit intNote >= 14
        MsgBox “Très bien”
End If
```

↯ *Select Case :*

Cette structure permet d’exécuter un bloc d’instructions selon la valeur d’une expression. L’expression peut être de tout type (chaîne, numérique, booléen..). On détermine ensuite le bloc à exécuter en définissant des cas exprimés par les valeurs que peut prendre l’expression.

La syntaxe est la suivante :

```
Select Case expression          ‘Expression à évaluer
Case liste expressions1
        Instructions
        …
Case liste expression2
        Instructions
        …
Case else                               ‘Cas facultatif ; aucun des autres
        Instructions                    ‘n’est vérifié
        …
End Select
```

Les expressions contenant les valeurs sont séparées par des virgules. Certaines définissent des intervalles *A To B* ou utilisent des opérateurs relationnels introduit par *Is* (Is > 15).

Exemple :

```
Dim intNote As Integer
intNote = 13

Select Case intNote             ‘Expression à évaluer
Case Is < 10                    ‘Cas où intNote est inférieur à 10
        MsgBox “Recalé”
Case 10 To 11
        MsgBox “Passable”
Case 12, 13
        MsgBox “Bien”
Case else
        MsgBox “Très bien”
End Select
```

↯ *Iif() : (Immediate if)*

Cette fonction renvoie le deuxième paramètre si l’expression du premier paramètre est vraie. Le troisième paramètre est retourné dans le cas contraire.

Condition, si vrai, si faux.

```
Iif(n < 10, "Ajourné", "Admis")
```

↳ *Switch() :*

Renvoie la valeur associée à la première condition vérifiée.

```
Switch(n<10, "Ajourné", n<12, "Admis", n<14, "Assez bien")
```

Dans ce cas, les paramètres fonctionnent deux à deux : une condition et une valeur. La fonction parcourt ces paramètres de gauche à droite en évaluant chaque condition. Dès qu'elle rencontre une condition vraie, elle renvoie la valeur qui suit la condition.
Dans ce cas, si « *n* » vaut 11, la fonction retourne « Admis ».

3.6.2 Les boucles

Il existe 3 types de boucles en VB. Ces structures permettent de répéter une ou plusieurs instructions. L'arrêt d'une boucle dépend de la valeur d'une expression booléenne ou d'un compteur.

Il est impératif de correctement définir les conditions de sortie d'une boucle, afin d'éviter une boucle infinie.

↳ *Do…Loop :*

Cette boucle peut prendre deux variantes selon la condition de sortie. Il peut s'agir d'un test de continuité (tant que, soit while), ou d'un test d'arrêt (jusqu'à ce que, soit until).
On utlise plutôt ce type de boucle lorsque l'on ne connaît pas à l'avance le nombre de tours à effectuer (recherche dans un tableau de données, remplissage d'une liste déroulante avec des données lues dans une table de base de données…)

```
Do [While|Until] condition
        Instructions
        Exit Do
Loop
```

Exemple :

```
Do While i < 5          'Tant que i est inférieur à 5
        Instruction
        i = i + 1       'Faire avancer le compteur
Loop
```

Dans le cas précédent, le test de sortie est situé au début de la boucle. Il se peut que le corps de la boucle ne soit jamais exécuté si la condition n'est pas vérifiée.
On peut alors placer le test de sortie à la fin de la boucle. Dans ce cas, le corps de la boucle est toujours exécuté au moins une fois.

```
Do
        Instructions
        Exit Do
Loop [While|Until] condition
```

Exit Do : cette commande permet de sortir de la boucle avant que la condition de sortie ne soit remplie. Cela peut être utile lors d'une recherche dans un tableau ou un jeu de données pour sortir de la boucle dès que l'on trouve la valeur recherchée.

Exemple :

```
Do While i < Ubound(Montab)              'Tant que i est inférieur à 5
        If Montab(i) = "Toto" Then
                Exit Do
        End If
        i = i + 1           'Faire avancer le compteur
Loop
```

↳ *For…Next :*

Ce type de boucle est utile lorsqu'on connaît le nombre de tour à effectuer. La variable *compteur* évolue de *début* à *fin* selon le *pas* défini.

```
For compteur = début To fin [Step pas]
        Instructions
        Exit For
Next compteur
```

Exemple :

```
For i = 0 To 10 Step 2
        Instructions
Next i
```

Comme précédemment, il existe une instruction permettant d'interrompre la boucle avant d'atteindre la valeur du paramètre *fin*. **Il n'est pas nécessaire de faire avancer explicitement le compteur car ce rôle est assuré par l'instruction Next.**

Le *pas* par défaut est 1. Il n'est pas nécessaire de le préciser dans ce cas. Il peut être négatif. Il faut alors que *Début* soit supérieur à *fin*.

↳ *For…Each :*

Ce type de boucle permet de parcourir une collection d'objets de même type ou une série d'objets contenus dans un objet conteneur. C'est le cas des contrôles dans un formulaire. Ils peuvent être issus de classes différentes, mais ils sont dans une collection *Controls*.

Remarque : lorsque l'on parcourt des objets de classes différentes, il faut veiller à n'accéder qu'à des propriétés ou méthodes communes à tous les objets, sinon une erreur d'exécution se produira.

```
For Each membre In collection
        Instruction
        Exit For
Next membre

Dim CtrlVar As Control
```

```
For Each CtrlVar In Form1.Controls
        MsgBox CtrlVar.Name             'Tous les contrôles ont une
                                                'propriété Name
Next CtrlVar
```

La récursivité :

Cela consiste à faire qu'une fonction s'appelle elle-même. Cela permet de parcourir des arbres dont on ne connaît pas la structure à l'avance. On peut ainsi traiter tous les cas. Il faut prendre garde aux conditions de sortie au risque de saturer la pile d'exécution.
Cette méthode peut remplacer des boucles, mais elle est plus lente. Toutefois, certains cas ne peuvent pas être traités par des boucles.

```
Function Facto(i as Integer) as Integer
        If i = 1 Then
                Facto = 1
        Else
                Facto = i * Facto(i – 1)
        End If
End Function
```

Etiquettes :

Les étiquettes permettent de sauter un groupe d'instructions. On utilise `Goto` pour atteindre une étiquette. Cette méthode est surtout utilisée dans la gestion des erreurs. Pour créer une étiquette, il suffit de lui donner un nom suivi de « : ».

```
Goto Toto
        Instructions
Toto :
        Instructions
```

Le recours à cette méthode est vivement déconseillé et doit être réservée à la gestion d'erreur (chapitre VI).

3.6.3 Les principales fonctions intégrées

Voir document annexe : « Fonctions intégrées.pdf. »

3.6.4 Manipulation de chaînes de caractères

De nombreuses fonctions intégrées à VB permettent de manipuler les chaînes de caractères. On retrouve une grande partie de ces fonctions dans d'autres langages.

(Les termes entre crochets sont facultatifs).

UCase(ExpressionChaine) : passe ExpressionChaine en majuscule.

```
        UCase("Toto") ➔ TOTO
```

Lcase(ExpressionChaine) : passe ExpressionChaine en minuscule

```
        Lcase("Toto") ➔ toto
```

Len(ExpressionChaine) : renvoie la longueur en caractère de ExpressionChaine.

```
Len("Toto") ➔ 4
```

Left(ExpressionChaine,NbCar) : renvoie NbCar de ExpressionChaine en partant de la gauche.

```
Left("Hello World", 7) ➔ Hello W
```

Right(ExpressionChaine,NbCar) :renvoie NbCar de ExpressionChaine en partant de la droite.

```
Right("Hello World", 7) ➔ o World
```

Mid(ExpressionChaine,Debut,[NbCar]) : renvoie NbCar de ExpressionChaine en partant de Debut. Si NbCar est omis, la fonction lit ExpressionChaine jusqu'à la fin.

```
Mid("Hello World", 3, 6) ➔ llo Wo
```

Ltrim(ExpressionChaine) : supprime les espaces à gauche de ExpressionChaine.

```
Ltrim("  Toto  ") ➔ "Toto  "
```

Rtrim(ExpressionChaine) : supprime les espaces à droite de ExpressionChaine.

```
Rtrim("  Toto  ") ➔ "  Toto"
```

Trim(ExpressionChaine) : supprime les espaces à droite et à gauche de ExpressionChaine.

```
Trim("  Toto  ") ➔ "Toto"
```

InStr([Debut], ExpressionChaine, ChaineCherchee, [Comparaison]) : renvoie la position de ChaineCherchee dans ExpressionChaine. La recherche commence à partir de Debut, mais renvoie la position à partir du premier caractère de la chaîne, selon une Comparaison binaire ou texte. La comparaison binaire (par défaut) prend en compte la casse, contrairement à la comparaison texte.

```
InStr(2, "Hello World", "W", 0) ➔ 7 (binaire)
InStr(2, "Hello World", "w", 0) ➔ 0 (binaire)
InStr(2, "Hello World", "w", 1) ➔ 7 (texte)
```

L'essentiel

Maitriser les déclarations de variables et les différentes portées possibles. Il est important de rendre obligatoire la déclaration.

Déclaration des procédures et des fonctions ainsi que leur portée. Comprendre l'intérêt et l'utilisation des paramètres.

Maitriser les structures de contrôle sur le plan algorithmique et syntaxique.