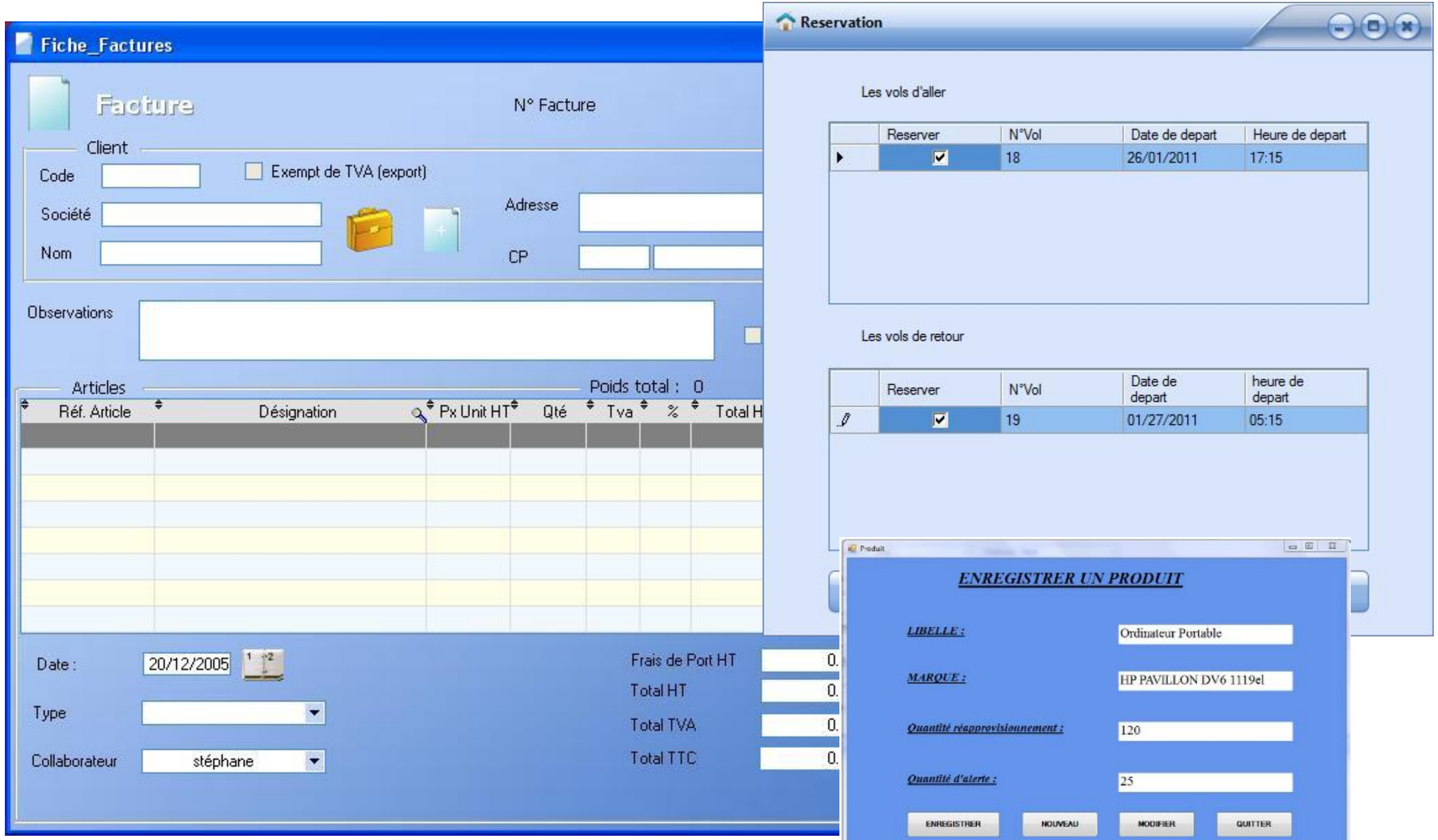
UNIVERSITÉ CHOUAIB DOUKKALI
Ecole Supérieure de Technologie
Sidi Bennour

Cours: Programmation Événementielle "VB.net"

Pr. B. CHERKAOUI


Fiche_Factures
Facture
N° Facture
Client
Code
Exempt de TVA (export)
Société
Nom
Adresse
CP
Observations
Articles
Poids total : 0
Réf. Article
Désignation
Px Unit HT
Qté
Tva
%
Total H
Date :
20/12/2005
Type
Collaborateur
stéphane
Frais de Port HT
Total HT
Total TVA
Total TTC
Reservation
Les vols d'aller
Reserver
N°Vol
Date de depart
Heure de depart
18
26/01/2011
17:15
Les vols de retour
Reserver
N°Vol
Date de depart
heure de depart
19
01/27/2011
05:15
Produit
ENREGISTRER UN PRODUIT
LIBELLE :
Ordinateur Portable
MARQUE :
HP PAVILLON DV6 1119el
Quantité réapprovisionnement :
120
Quantité d'alerte :
25
ENREGISTRER
NOUVEAU
MODIFIER
QUITTER

Historique, naissance du Visual Basic
D'où vient le Visual Basic ?

Le BASIC

✓ BASIC est un acronyme pour **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. Le BASIC a été conçu en 1963 par John George Kemeny et Thomas Eugene Kurtz pour permettre aux étudiants qui ne travaillaient pas dans des filières scientifiques d'utiliser les ordinateurs.

✓ **Les huit principes de conception du BASIC étaient** :

- Être facile d'utilisation pour les débutants (Beginner) ;
- Être un langage généraliste (All-purpose) ;
- Autoriser l'ajout de fonctionnalités pour les experts (tout en gardant le langage simple pour les débutants) ;
- Être interactif ;
- Fournir des messages d'erreur clairs et conviviaux ;
- Avoir un délai de réaction faible pour les petits programmes ;
- Isoler l'utilisateur du système d'exploitation.

Historique, naissance du Visual Basic

D'où vient le Visual Basic ?

✓ Le Visual Basic:

✓ De ce langage — le BASIC — est né le Visual Basic. Le VB est directement dérivé du BASIC et permet le développement rapide d'applications, la création d'interfaces utilisateur graphiques, l'accès aux bases de données, ainsi que la création de contrôles ou d'objets ActiveX.

✓ Le traditionnel « **Hello World !** » en Visual Basic :

✓ **Code : Autre**

```
Sub Main()
        MsgBox("Hello World !")
End Sub
```

VB & VB.net

- le VB a laissé place au VB .NET. Le suffixe **.NET** spécifie en fait qu'il nécessite le *framework* **.NET** de Microsoft afin de pouvoir être exécuté. À savoir qu'il est également possible d'exécuter un programme créé en VB sous d'autres plates-formes que Windows grâce à Mono.
- **M'sieur… c'est quoi un *framework* ?**

Framework .NET

- Un framework (**dans notre cas, le framework .NET de Microsoft**) est une sorte d'immense bibliothèque informatique contenant des outils qui vont faciliter la vie du développeur. Le framework .NET est compatible avec le Visual Basic et d'autres langages tels que le C#, le F#, le J#, etc.

Le framework .NET

- La plate-forme .NET fournit un ensemble de fonctionnalités qui facilitent le développement de tous types d'applications :
  - **Les applications Windows classiques ;**
  - **Les applications web ;**
  - **Les services Windows ;**
  - **Les services web.**

En Visual Basic, toutes ces applications sont réalisables grâce au framework .NET.

Notre premier programme !

- Nous allons donc aborder les principes fondamentaux du langage. Pour cela, empressons-nous de créer un **nouveau projet**, cette fois en **application console**.
- **Évitez** d'utiliser des **accents ou caractères spéciaux** dans un nom de fichier ou de projet.
- Créez un nouveau projet (**CTRL + N**) et cliquez sur Application console.
- Voici ce qui devrait s'afficher chez vous :

```
Module Module1

    Sub Main()

    End Sub

End Module
```

Notre premier programme !

- Ces mots figurant dans votre feuille de code sont indispensables ! Si vous les supprimez, l'application ne se lancera pas. C'est le code minimal que l'IDE (Visual Studio) génère lorsque l'on crée un projet de type console.
- Chaque grosse partie, telle qu'une *fonction*, un *module*, un *sub*, voire une *boucle conditionnelle*, aura une balise de début : ici, **Module** Module1 et **Sub** Main(), et une balise de fin : **End Module** et **End Sub**.
- Module1 est le nom du module, que vous pouvez le modifié.
- Pour ce qui est du **Main ( )**, n'y touchez pas, car lorsqu'on va lancer le programme la première chose que ce dernier va faire sera de localiser la partie appelée Main() et de sauter dedans. S'il ne la trouve pas, cela ne fonctionnera pas !

Les « parties » telles que Main() sont appelées des méthodes, car elles sont précédées de Sub.

Notre premier programme « Hello World ! »

- Cette console vous permettra d'apprendre les bases et les concepts fondamentaux du VB sans vous embrouiller directement l'esprit avec les objets qui orneront nos interfaces graphiques. Nous allons donc créer un programme qui écrit dans cette console.

Hello World !

Code : VB.NET

```
Module Module1
    Sub Main()
        Console.Write("Hello World !")
    End Sub
End Module
```

---> Cette ligne est appelée une **instruction**.

Déroulement du programme

- Le programme entre dans le Main() et exécute les actions de haut en bas, instruction par instruction. Attention, ce ne sera plus le cas lorsque nous aborderons des notions telles que les boucles ou les fonctions.
- Voici nos lignes de code :
  1. Module Module1 : le programme entre dans son module au lancement. Forcément, sinon rien ne se lancerait jamais. La console s'initialise donc.
  2. Il se retrouve à entrer dans le Main(). La console est ouverte.
  3. Il continue et tombe sur notre ligne qui lui dit « **Affiche « Hello World !"** », il affiche donc « Hello World ! » dans la console.
  4. Il arrive à la fin du Main() (End Sub). Rien ne se passe, « Hello World ! » est toujours affiché.
  5. Il rencontre le End Module : la console se ferme.
- Résultat des courses : la console s'est ouverte, a affiché « Hello World ! » et s'est fermée à nouveau… mais tout cela en une fraction de seconde, on n'a donc rien remarqué !

Notre premier programme avec *une pause*

- La parade : donner au programme une ligne à exécuter sur laquelle il va attendre quelque chose. On pourrait bien lui dire attendre une entrée. Oui, la touche Entrée de votre clavier.
- Pour cela, voici la ligne de code qui effectue cette action :

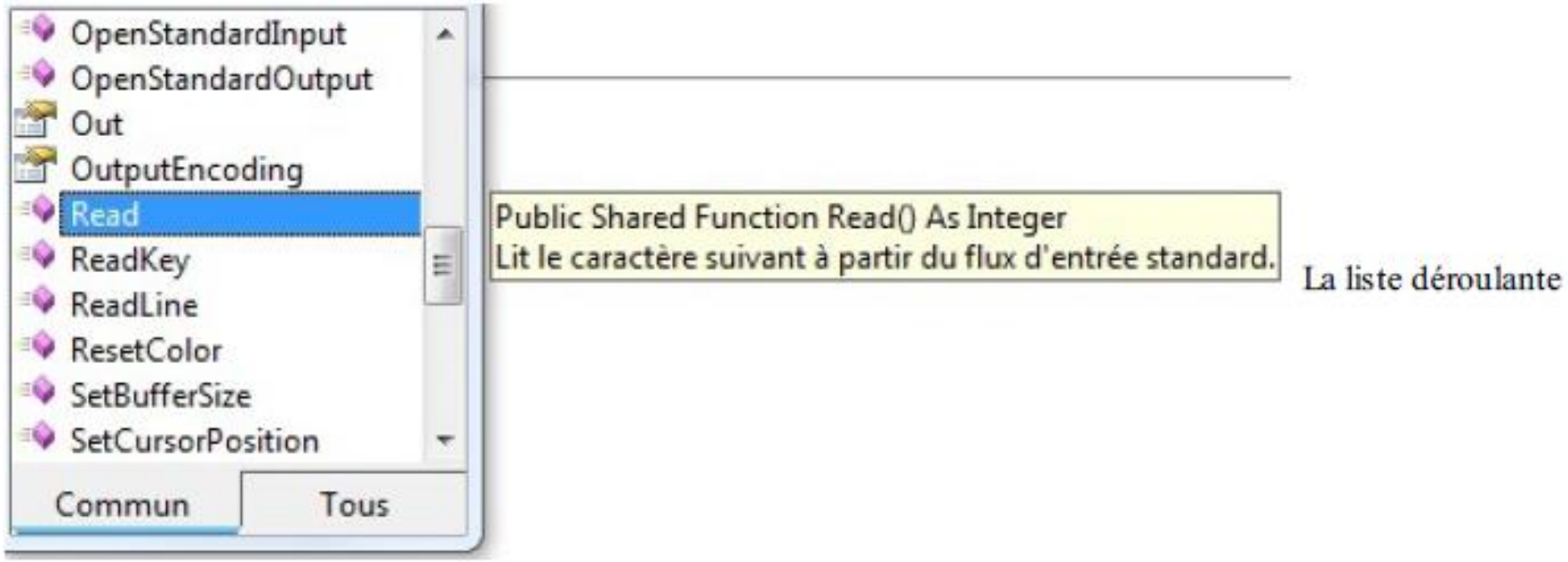
Code : VB.NET

```
Module Module1
    Sub Main()
        Console.Write("Hello World !")
        Console.Read()
    End Sub
End Module
```

- Cette ligne dit à l'origine « Lis le caractère que j'ai entré », mais nous allons l'utiliser pour dire au programme : « Attends l'appui sur la touche Entrée ».

Objets, fonctions…

- Vous l'avez peut-être remarqué : au moment où vous avez écrit « Console. », une liste s'est affichée en dessous de votre curseur (voir figure suivante). Dans cette partie, je vais vous expliquer l'utilité de cette liste.



La liste déroulante

Fonctions

- Une fonction est une séquence de code déjà existante et conçue pour obtenir un effet bien défini. Concrètement, cela nous permet de n'écrire qu'une seule fois ce que va faire cette séquence, puis d'appeler la fonction correspondante autant de fois que nous le voulons (la séquence exécutera bien entendu ce qu'on a défini au préalable dans la fonction…).
- Par exemple, nos deux lignes qui nous permettaient d'afficher « *Hello World !* » et d'effectuer une pause auraient pu être placées dans une fonction séparée. Dans ce cas, en une ligne (l'appel de la fonction), on aurait pu effectuer cette séquence.
- Alors le gain de temps et les avantages dans des séquences de plusieurs centaines de lignes.
- Un autre exemple : notre fonction ***Write*** avait pour but d'écrire ce que l'on lui donnait comme **arguments**. La fonction ***Write*** a donc été écrite par un développeur qui y a placé une série d'instructions permettant d'afficher du texte dans la console.

Fonctions, Arguments

- Pas de panique si vous n'avez pas compris ces concepts de fonctions, d'objets, etc. Nous allons justement nous pencher sur la structure d'un appel de fonction, car nous en aurons besoin très bientôt ; pour cela, nous allons étudier une fonction simple : le ***BEEP*** (pour faire « bip » avec le haut-parleur de l'ordinateur). Afin d'y accéder, nous allons écrire ***Console.Beep***.
- Ici, deux choix s'offrent à nous : le classique *()* ou alors (*frequency as integer, duration as integer*).

Exemple de fonctions avec arguments

- La première forme va émettre un « **bip** » classique lors de l'exécution.
- La seconde demande des arguments. Il s'agit de paramètres passés à la fonction pour lui donner des indications plus précises. Précédemment, lorsque nous avons écrit Write("**Hello World !**"), l'argument était "Hello World !" ; la fonction l'a récupéré et l'a affiché.
- Pour certaines fonctions, on a le choix de donner des arguments ou non.
- La seconde forme prend donc deux arguments, que vous voyez d'ailleurs s'afficher dès que vous tapez quelque chose entre les parenthèses, comme sur l'une des images ci-avant. Le premier sert à définir la fréquence du « bip » : entrez donc un nombre pour lui donner une fréquence. Le second, quant à lui, détermine la durée du « bip ».

Code : VB.NET

```
Console.Beep(500, 100)
```

- Pourquoi n'y a-t-il pas de guillemets (doubles quotes : « " ») autour des nombres ?

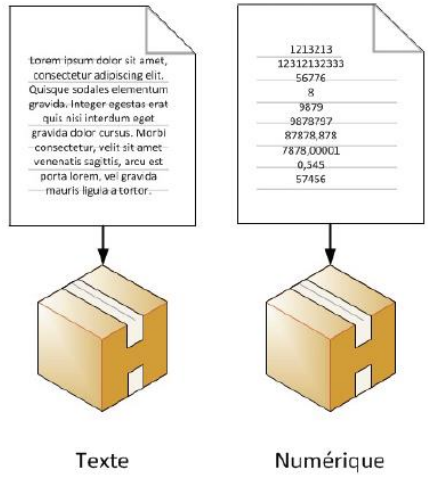
Les variables

- Comme son nom l'indique, une variable… varie. On peut y stocker pratiquement tout ce qu'on veut, comme par exemple des nombres, des phrases, des tableaux, etc. C'est pour cette raison que les variables sont omniprésentes dans les programmes.

Types de variables:

- Les variables se déclinent sous différents types : il y a par exemple un type spécifique pour stocker des nombres, un autre pour stocker du texte, etc.

Nom	Explication
`Boolean`	Ce type n'accepte que deux valeurs : vrai ou faux. Il ne sert à rien, me direz-vous ! détrompez-vous.
`Integer`	Type de variable spécifique au stockage de nombres entiers (existe sous trois déclinaisons ayant chacune une quantité de « place » différente des autres).
`Double`	Stocke des nombres à virgule.
`String`	Conçu pour stocker des textes ou des mots. Peut aussi contenir des nombres.
`Date`	Stocke une date et son heure sous la forme « 12/06/2009 11:10:20 ».



Les variables : *Le Sens*

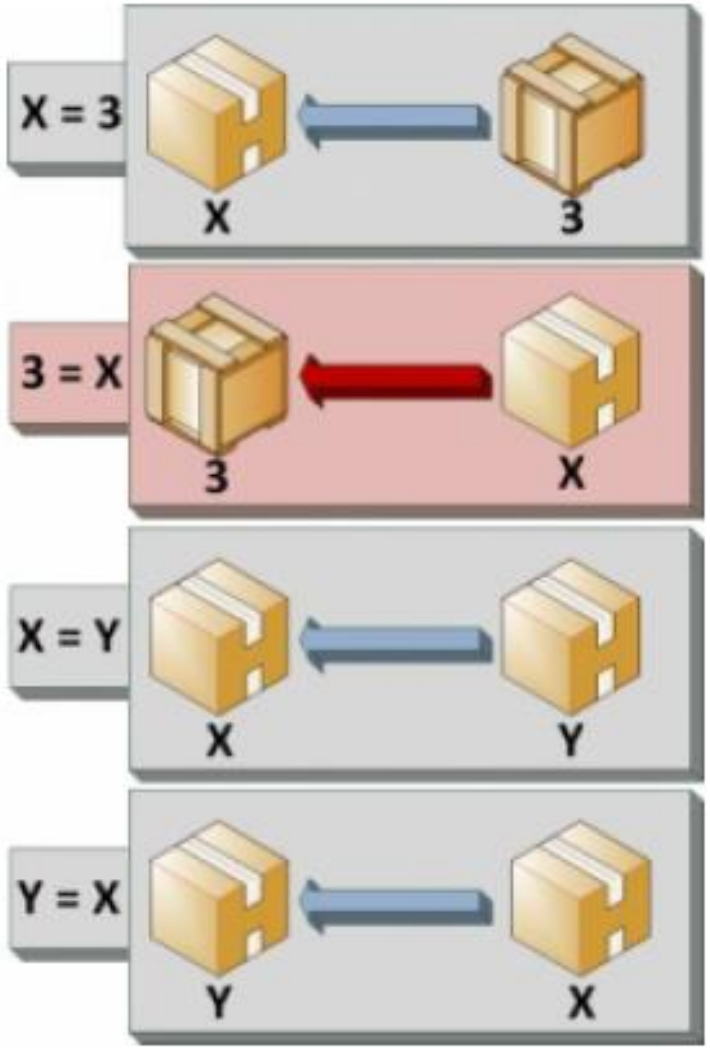
- En VB, et même dans tous les langages de programmation, ce qui se situe à gauche du « = » correspond à l'opération qui se trouve à droite.

1. On entre le chiffre 3 dans la variable appelée X, pas de problème ;

2. Ensuite, on souhaite mettre X dans 3 ! Aïe, cela ne va pas fonctionner ! Si vous écrivez quelque chose de ce genre, une erreur va se produire : comme si vous disiez « 3 = 2 », le compilateur va vous regarder avec des yeux grands comme ça et se demandera ce qu'il doit faire !
3. Ensuite, on met la variable Y dans la variable X ;

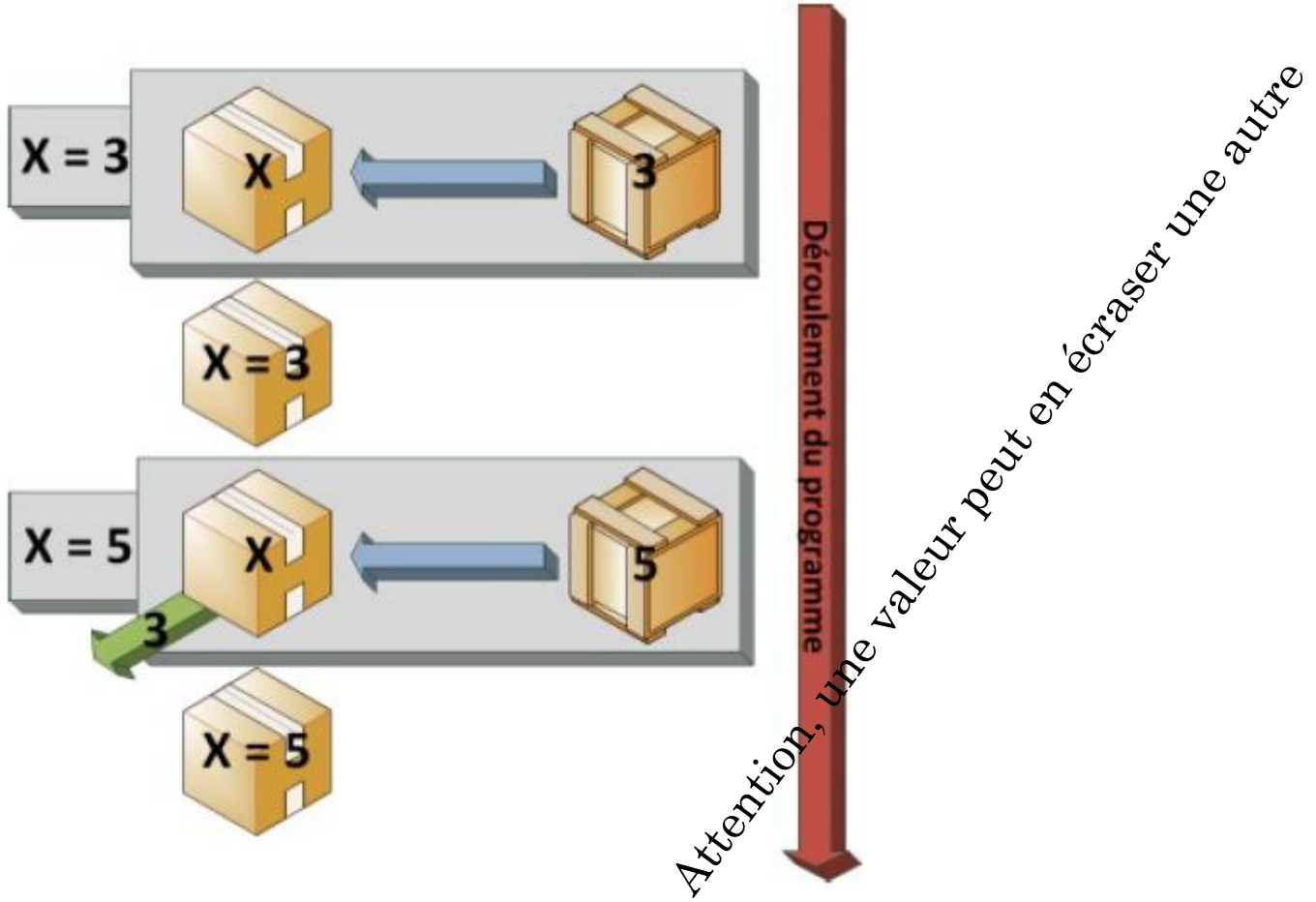
4. Et enfin, X dans Y.

Affectations de valeurs à des variables



Les variables : *Le Sens*

- L'affectation d'une valeur à une variable écrase l'ancienne valeur, comme schématisé à la figure suivante.



Code : VB.NET

```
Dim MaVariable As Integer
        MaVariable = 5
        MaVariable = 8
        MaVariable = 15
        MaVariable = 2
        MaVariable = 88
        MaVariable = 23
```

Que vaudra MaVariable à la fin de ces instructions ?

Déclaration de variable

- Déclarer une variable, par exemple de type ***integer*** :

  Code : VB.NET

```
Dim MaVariable As Integer
```

Code VB	Dim	MaVariable	As	Integer
Français	Crée une variable	de nom « MaVariable »	en tant que	*entier*

le mot « MaVariable » est le nom attribué à la variable. C'est vous qui le choisissez !

Le nom d'une variable ne peut contenir d'espaces ; privilégiez plutôt un « _ » (underscore) ou une majuscule à chaque « nouveau mot », mais en liant le tout (comme dans mon exemple).

Autre chose à propos des noms : il y a des exceptions. En effet, une variable ne peut pas avoir comme nom un type ou le nom d'une boucle. Par exemple, si vous appelez votre variable « Date », une erreur se produira, car le type Date existe déjà.

Exemple de déclaration des variables

1. Créer une variable de type ***String*** appelée « ***StringFunction*** » et d'y entrer la valeur « **Bonjour tout le monde** »
2. Créer une variable de type ***Integer*** appelée « ***MaVariable*** » et d'y entrer la valeur « ***5*** »
3. Afficher les deux variables?

Modifications des variables et opérations sur les variables

Opérations sur les variables

Déclarer une variable **MaVariable1** en **Integer** et assignons-lui la valeur 5, puis Déclarer une seconde variable intitulée **MaVariable2**, de nouveau en **Integer**, et assignons-lui cette fois la valeur 0.

Code : VB.NET

```
Module Module1
    Sub Main()
        Dim MaVariable1 As Integer
        Dim MaVariable2 As Integer
        MaVariable1 = 5
        MaVariable2 = 0
        Console.Write(MaVariable)
        Console.Read()
    End Sub
End Module
```

✓ Pour la déclaration de plusieurs variables de même type:

```
Dim MaVariable1, MaVariable2 As Integer
```

✓ Pour initialiser vos variables dès leur déclaration

```
Dim MaVariable As Integer = 5
```

Opérateurs arithmétiques & Opérateurs de comparaison

Opération souhaitée	Symbole
Addition	+
Soustraction	-
Multiplication	*
Division	/
Division entière	\
Puissance	^
Modulo	Mod

Dans l'algèbre standard	Equivalent dans Visual Basic
$a = b$	a = b
$c \neq d$	c <> d
$e > f$	e > f
$g < h$	g < h
$i \geq j$	i >= j
$k \leq l$	k <= l

Exemple
Code : VB.NET

```
Module Module1
    Sub Main()
        Dim var1, var2, var3 As Integer
        var1 = 10
        var2 = 20
        var3 = var1 + var2
        Console.Write(var3)
        Console.WriteLine()
        Console.WriteLine("20 Modulo 10 est egal = " & 20 Mod 10)
        Console.WriteLine(" Multiplication de 20 * 10 est = " & var1 * var2)
        Console.Read()
    End Sub
End Module
```

La **concaténation**, elle permet d'assembler deux éléments en un ; ici, par exemple, elle a assemblé la chaîne de caractères " `20 Modulo 10 est egal =` " et le contenu de la variable, ce qui aura pour effet de m'afficher directement 20 Modulo 10 est egal = 0

Les commentaires

- Les commentaires vont nous servir à éclaircir le code. Ce sont des phrases ou des indications que le programmeur laisse pour lui même ou pour ceux qui travaillent avec lui sur le même code.
- Une ligne est considérée comme commentée si le caractère « ' » (autrement dit, une simple quote) la précède ; une ligne peut aussi n'être commentée qu'à un certain niveau.

Exemples :

Code : VB.NET

```
'Commentaire
MaVariable = 9 * 6 ' Multiplie 9 et 6 et entre le résultat dans MaVariable
```

Les commentaires

- Par exemple, voici notre programme dûment commenté :
- **Code : VB.NET**

```
Module Module1
    Sub Main()
        'Initialisation des variables
        Dim MaVariable As Integer = 8
        Dim MaVariable2 As Integer = 9

        'Affiche "9 x 8 = " puis le résultat (multiplication de MaVariable par MaVariable2)
        Console.Write("9 x 8 = " & MaVariable * MaVariable2)

        'Crée une pause factice de la console
        Console.Read()
    End Sub
End Module
```

Conditions et boucles conditionnelles

- Une boucle conditionnelle est quelque chose de fort utile et courant en programmation. Cela permet d'effectuer une action si, et seulement si, une condition est vérifiée.
- Par exemple vous voulez que votre programme dise « bonne nuit » s 'il est entre 22 h et 6h. Eh bien, c'est précisément dans ce cas de figure que les boucles conditionnelles trouvent leur utilité.

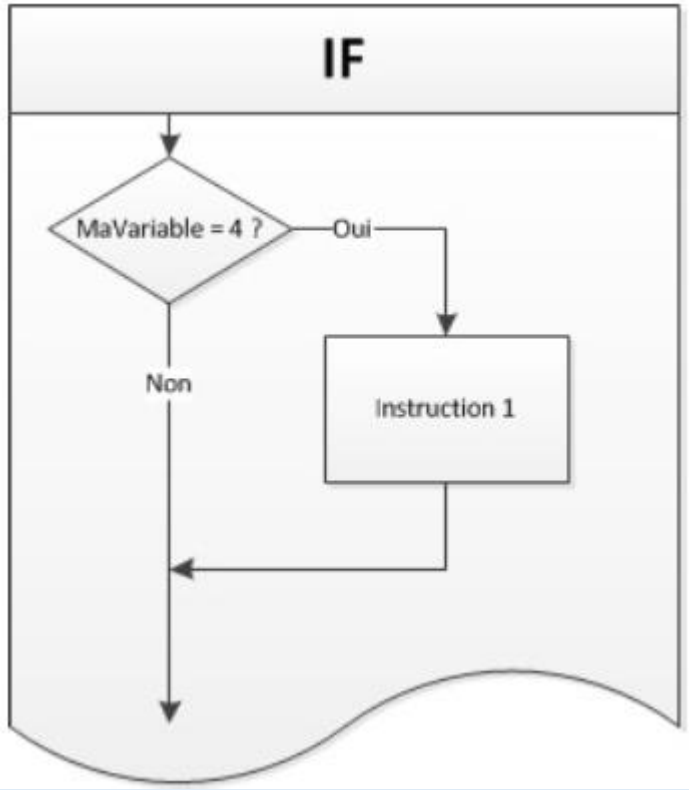
Conditions et boucles conditionnelles

Boucle If

- Une ligne commençant par **`If` est toujours terminée par `Then`, ce qui signifie « Si, alors ». C'est entre ces deux mots que vous** placez la condition souhaitée.
- Donc, si j'écris le code **`If MaVariable = 10 Then`, ce qui se trouve en dessous ne sera exécuté que si la valeur de** `MaVariable` est égale à 10. Regardez la figure suivante.

Eh bien oui, du moins jusqu'à ce qu'il rencontre **`End If`, traduisible par « Fin si ». Comme pour un `Sub` ou un `Module`, une** boucle est associée à sa fin correspondante.
En clair, **`If`, `Then` et `End If` sont indissociables !**

Code VB	`If`	`MaVariable`	`= 10`	`Then`
Français	Si	« MaVariable »	est égale à 10	alors



Conditions et boucles conditionnelles

Boucle If

- **Exemple:**

Code : VB.NET

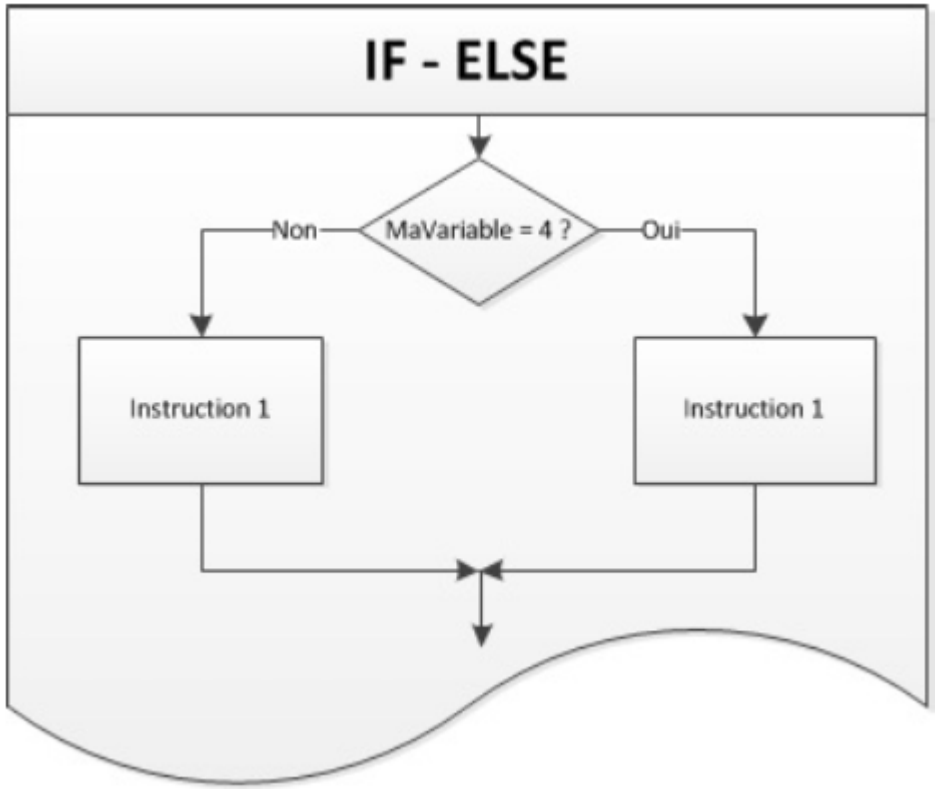
```
If Moyen = 12 Then
       Module = "Valide"
End If
```

- Si `Moyenne` est égale à 12, il met `Module = "Valide"`.

Conditions et boucles conditionnelles

- ***Boucle if Else***

« Else », il faut y penser parfois pour gérer toutes les éventualités.



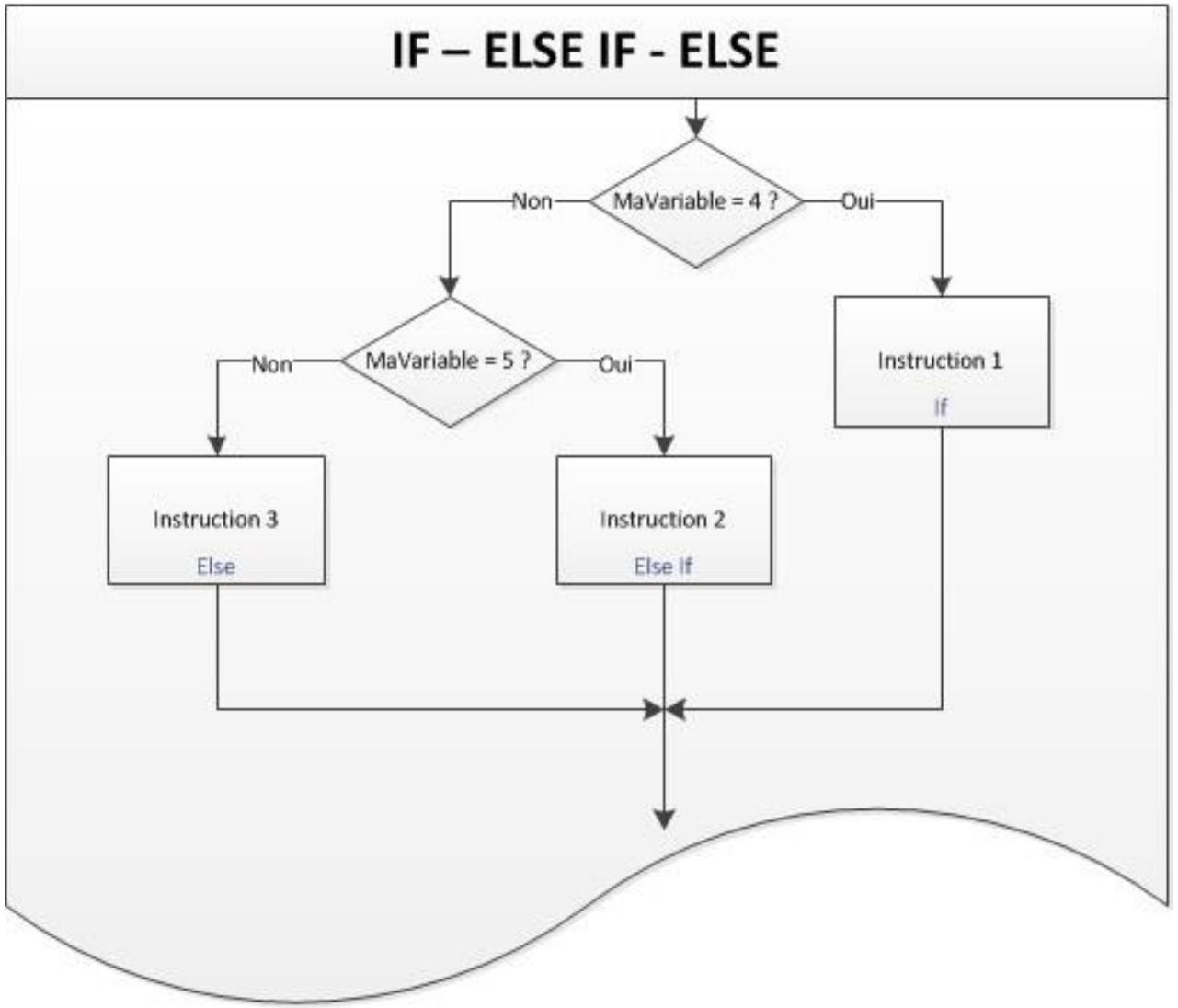
La syntaxe est la suivante :
Code : VB.NET

```
If Moyen= 12 Then
    'Code exécuté si Moyen = 12
Else
    'Code exécuté si Moyen est infèrieure à 12
End If
```

Conditions et boucles conditionnelles

- **La Boucle *ElseIf***

La figure suivante schématise le **ElseIf**.



- Voici un exemple :

Code : VB.NET

```
If MaVariable = 10 Then
    'Code exécuté si MaVariable = 10
ElseIf MaVariable = 5 Then
    'Code exécuté si MaVariable = 5
Else
    'Code exécuté si MaVariable est
différente de 10 et de 5
End If
```

Conditions et boucles conditionnelles

- les boucles **If**, **Then** et **ElseIf** peuvent ***s'imbriquer, ce qui signifie qu'on peut en mettre plusieurs*** l'une dans l'autre. Contrairement à **If** et **ElseIf**, le **Else** ne peut être placé qu'une seule et unique fois dans une condition.

Code : VB.NET

```
If MaVariable = 10 Then
    If MaVariable2 = 1 Then
        'Code exécuté si MaVariable = 10 et MaVariable2 = 1
    Else
        'Code exécuté si MaVariable = 10 et MaVariable2 <> 1
    End If
ElseIf MaVariable = 5 Then
    If MaVariable2 = 2 Then
        'Code exécuté si MaVariable = 5 et MaVariable2 = 2
    End If
Else
        'Code exécuté si MaVariable est différente de 10 et de 5
End If
```

Conditions et boucles conditionnelles

- **Select**
  - ✓ Nous avons vu **`If`**, **`ElseIf`** et **`Else`**.
  - ✓ Mais pour ce qui est, par exemple, du cas d'un menu dans lequel vous avez 10 choix différents, comment faire ?
  - ✓ Une première façon de procéder serait la suivante :

✓ **Code : VB.NET**

```
If Choix = 1 Then
    Console.WriteLine("Vous avez choisi le menu n° 1")
    ElseIf Choix = 2 Then
        Console.WriteLine("Vous avez choisi le menu n° 2")
        ElseIf Choix = 3 Then
            Console.WriteLine("Vous avez choisi le menu n° 3")
            ElseIf Choix = 4 Then
                Console.WriteLine("Vous avez choisi le menu n° 4")
            ElseIf Choix = 5 Then
                Console.WriteLine("Vous avez choisi le menu n° 5")
            ElseIf Choix = 6 Then
                Console.WriteLine("Vous avez choisi le menu n° 6")
            ElseIf Choix = 7 Then
                Console.WriteLine("Vous avez choisi le menu n° 7")
            ElseIf Choix = 8 Then
                Console.WriteLine("Vous avez choisi le menu n° 8")
            ElseIf Choix = 9 Then
                Console.WriteLine("Vous avez choisi le menu n° 9")
            ElseIf Choix = 10 Then
                Console.WriteLine("Vous avez choisi le menu n° 10")
            Else
                Console.WriteLine("Le menu n'existe pas")
End If
```

Conditions et boucles conditionnelles

- **La Boucle Select**
- Il faut néanmoins que vous sachiez que les programmeurs sont très fainéants, et ils ont trouvé sans cesse des moyens de se simplifier la vie. C'est donc dans le cas que nous venons d'évoquer que les Select **deviennent indispensables, et grâce** auxquels on simplifie le tout. La syntaxe se construit de la manière suivante :

Code VB	Select	Case	MaVariable
Français	Sélectionne	dans quel cas	« MaVariable » vaut

Code : VB.NET

```
Select Case MaVariable
       Case 1
            'Si MaVariable = 1
       Case 2
            'Si MaVariable = 2
 Case Else
            'Si MaVariable <> 1 et <> 2
   End Select
```

Conditions et boucles conditionnelles

- Dans le même cas de figure, revoici notre menu :

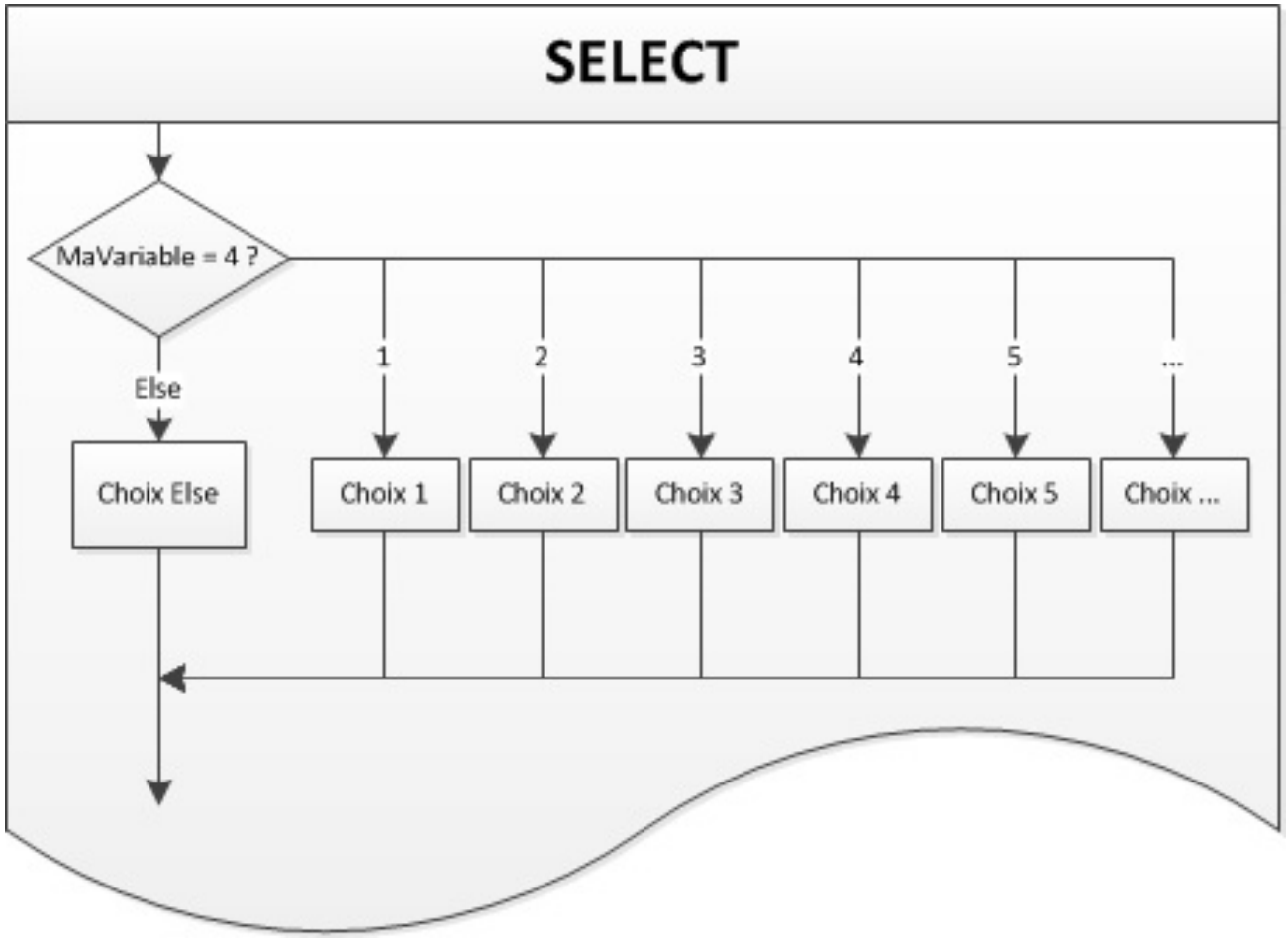
Code : VB.NET

```
Select Case Choix
    Case 1
        Console.WriteLine("Vous avez choisi le menu n° 1")
    Case 2
        Console.WriteLine("Vous avez choisi le menu n° 2")
    Case 3
        Console.WriteLine("Vous avez choisi le menu n° 3")
    Case 4
        Console.WriteLine("Vous avez choisi le menu n° 4")
    Case 5
        Console.WriteLine("Vous avez choisi le menu n° 5")
    Case 6
        Console.WriteLine("Vous avez choisi le menu n° 6")
    Case 7
        Console.WriteLine("Vous avez choisi le menu n° 7")
    Case 8
        Console.WriteLine("Vous avez choisi le menu n° 8")
    Case 9
        Console.WriteLine("Vous avez choisi le menu n° 9")
    Case 10
        Console.WriteLine("Vous avez choisi le menu n° 10")
    Case Else
        Console.WriteLine("Le menu n'existe pas")
End Select
```

Conditions et boucles conditionnelles

- Voici un petit schéma en figure suivante.

Fonctionnement de Select



Conditions et boucles conditionnelles

▪ *astuces avec* `Select`

✓ Si je souhaite que pour les valeurs 3, 4 et 5 il se passe la même action, dois-je écrire trois **`Case` avec la même instruction** ?

✓ Non, une petite astuce du **`Select` est de rassembler toutes les valeurs en un seul `Case`. Par exemple, le code suivant…**

Code : VB.NET

```
Select Case Choix
       Case 3,4,5
            'Choix 3, 4 et 5
End Select
```

✓ … est identique à celui-ci :

Code : VB.NET

```
Select Case Choix
       Case 3
            'Choix 3, 4 et 5
       Case 4
            'Choix 3, 4 et 5
       Case 5
            'Choix 3, 4 et 5
End Select
```

Conditions et boucles conditionnelles

- ***astuces avec `Select`***

✓ Astuce également valable pour de grands intervalles : le code suivant…

✓ **Code : VB.NET**

```
Select Case Choix
       Case 5 to 10
            'Choix 5 à 10
End Select
```

… correspond à ceci :

✓ **Code : VB.NET**

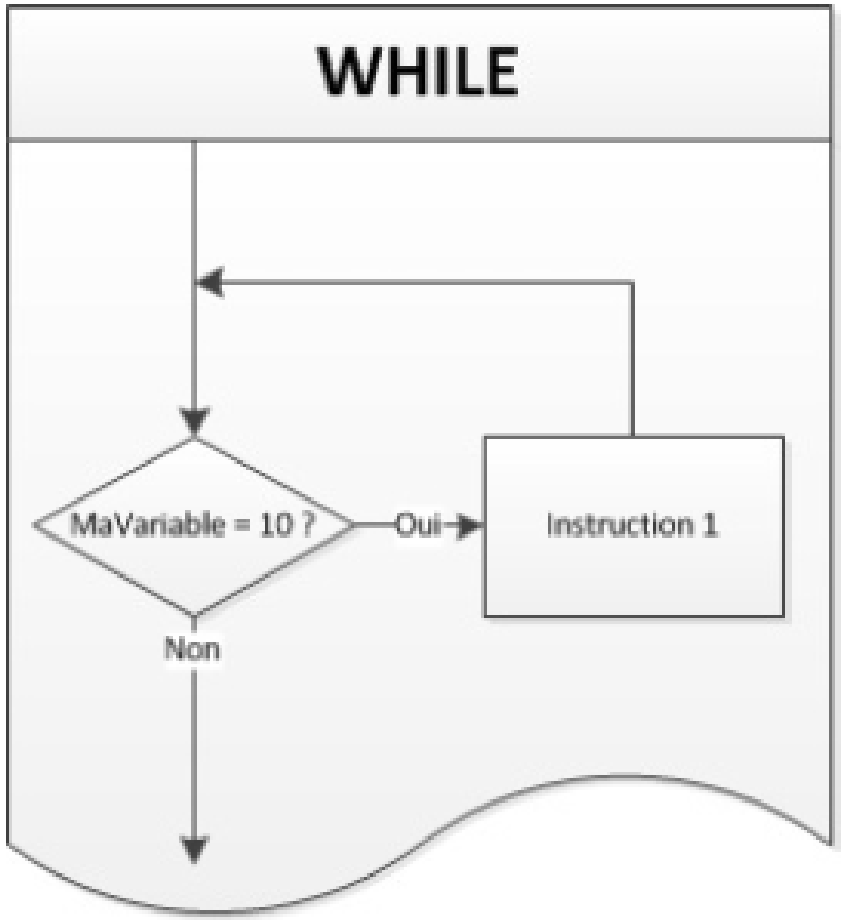
```
Select Case Choix
       Case 5
            'Choix 5 à 10
       Case 6
            'Choix 5 à 10
       Case 7
            'Choix 5 à 10
       Case 8
            'Choix 5 à 10
       Case 9
            'Choix 5 à 10
       Case 10
            'Choix 5 à 10
End Select
```

Conditions et boucles conditionnelles

- **While. « tant que »**
- Elle va effectivement « tourner » tant que la condition est **vraie.**
- La syntaxe est similaire à celle du `If.`
- **Code : VB.NET**

```
While MaVariable = 10
    'Exécuté tant que MaVariable = 10
End While
```

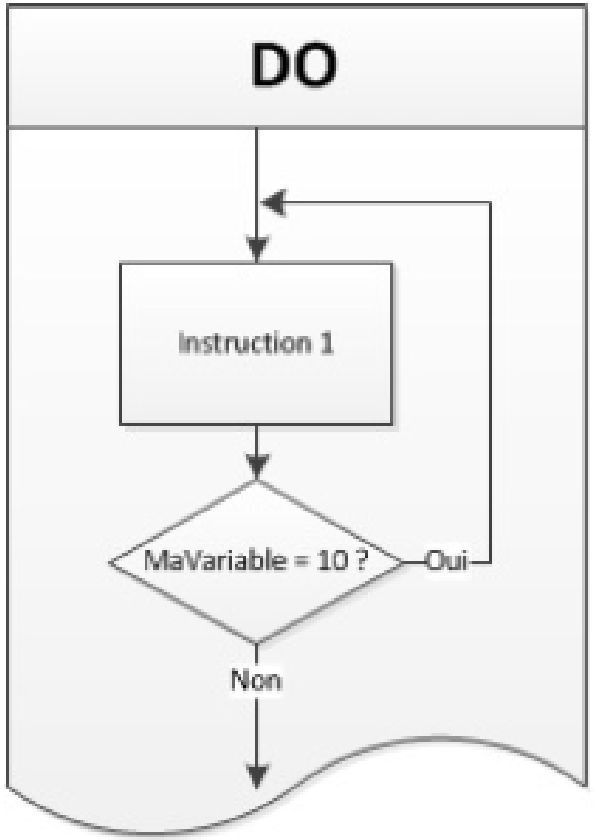
Code VB	While	MaVariable	= 10
Français	Tant que	« MaVariable »	est égale à 10



- En clair, le programme arrive au niveau de l'instruction While, vérifie que la condition est vraie et, si c'est le cas, entre dans le While, puis exécute les lignes qui se trouvent à l'intérieur ; il arrive ensuite au End While et retourne au While. Cela tant que la condition est vraie.

Do While

- À l'instar du **While**, **le Do While (traduisible par « faire tant que ») passe au moins une fois dans la boucle. Regardez la figure** suivante.



Code : VB.NET

```
Do
    'Instruction exécutée au moins une fois
Loop While MaVariable = 10
```

Autre information : il existe un autre mot qui se met à la place de « *While* ». *Ce mot est* « ***Until* ». *Il signifie :* « *passe*** tant que la condition n'est pas vraie » (le **While est utilisé seulement tant que la condition est vraie).**

Un code de ce type…

```
Do
Loop Until MaVariable = 10
```

… revient à écrire ceci :

Code : VB.NET

```
Do
Loop While MaVariable <> 10
```

Conditions et boucles conditionnelles

- **For « pour »**

✓ **For est indissociable de son To, comme un If a son Then**

✓ Si je souhaite effectuer une instruction dix fois de suite, je vais écrire ceci :

Code : VB.NET

```
Dim x As Integer = 0
While x <> 10
    'Instruction à exécuter 10 fois
    x = x + 1   'Augmente x de 1
End While
```

✓ La boucle sera parcourue à dix reprises. Eh bien, **For remplace ce code par celui-ci :**

Code : VB.NET

```
Dim x As Integer
For x = 1 to 10
    'Instruction à exécuter 10 fois
Next
```

Conditions et boucles conditionnelles

For

✓ Les deux codes effectueront la même chose. Le **Next** correspond à « ajoute 1 à ma variable ».

Code VB	For	MaVariable	= 1	To	10
Français	Pour	« MaVariable »	de 1	jusqu'à	10

✓ **Petites astuces du For…**

- On peut déclarer les variables dans la ligne du For, de cette manière :
- **Code : VB.NET**

```
For x As Integer = 1 to 10
    'Instruction à exécuter 10 fois
Next
```

- Cela reviendra de nouveau au même.
- Si vous voulez ajouter 2 au Next à la place de 1 (par défaut) :
- **Code : VB.NET**

```
For x As Integer = 1 to 10 step 2
    'Instruction à exécuter 5 fois
Next
```

Mieux comprendre et utiliser les boucles Opérateurs

- Pour valider la condition d'une boucle, il existe des opérateurs :

Symbole	Fonction
=	Égal
<>	Différent
>	Strictement supérieur
<	Strictement inférieur
<=	Inférieur ou égal
>=	Supérieur ou égal

- Exemple:
  - Si vous voulez exécuter un **`While`** **tant que « x » est plus petit que 10 :**
  - **Code : VB.NET**

```
While x < 10
```

Mieux comprendre et utiliser les boucles Opérateurs

- **And, or, not**
  - ❑ *Non, pas question !*
    - Commençons donc par le mot-clé **not**, **dont le rôle est de préciser à la boucle d'attendre l'inverse.**
    - Exemple : un **While not = 10 correspond à un While <> 10.**
  - ❑ *Et puis ?*
    - Un second mot permet d'ordonner à une boucle d'attendre plusieurs conditions : ce cher ami s 'appelle **And**. Il faut que toutes les
    - conditions reliées par **And** soient vérifiées.
    - **Code : VB.NET**

```
While MaVariable >= 0 And MaVariable <= 10
```

    - Ce code tournera tant que la variable est comprise entre 0 et 10.
    - Faites attention à rester logiques dans vos conditions :
    - **Code : VB.NET**

```
While MaVariable = 0 And MaVariable = 10
```

    - Le code précédent est totalement impossible, votre condition ne pourra donc jamais être vraie…

Mieux comprendre et utiliser les boucles Opérateurs

- **And, or, not**
  - ***Ou bien ?***
    - Ce mot permet de signifier « soit une condition, soit l'autre ».
    - Voici un exemple dans lequel **Or est impliqué :**
    - **Code : VB.NET**

```
While MaVariable >= 10 Or MaVariable = 0
```

    - Cette boucle sera exécutée tant que la variable est supérieure ou égale à 10, ou égale à 0.
- Ces mots peuvent s 'additionner, mais attention au sens.
- **Code : VB.NET**

```
While MaVariable > 0 And not MaVariable >= 10 Or MaVariable = 15
```

Ce code se comprend mieux avec des parenthèses : `(MaVariable > 0 et non MaVariable >= 10) ou MaVariable = 15`.
Donc, cela se traduit par « si `MaVariable` est comprise entre 1 et 10 ou si elle est égale à 15 ».

Jouer avec les mots, les dates

D'ici peu de temps vous pourrez modifier des mots aussi rapidement que des nombres.

Les chaînes de caractères

Remplacer des caractères

- On va commencer par la fonction la plus simple : le `Replace()` qui, comme son nom l'indique, permet de remplacer des
- caractères ou groupes de caractères au sein d'une chaîne.
- La syntaxe est la suivante :
- **Code : VB.NET**

```
Dim MonString As String = "Phrase de test"
MonString = MonString.Replace("test", "test2")
```

Le premier argument de cette fonction est le caractère (ou mot) à trouver, et le second, le caractère (ou mot) par lequel le remplacer.
Dans cette phrase, le code remplacera le mot « test » par « test2 ».
Si vous avez bien assimilé le principe des fonctions, des variables peuvent être utilisées à la place des chaînes de caractères en «dur ».

Jouer avec les mots, les dates

Les chaînes de caractères

- *Mettre en majuscules*
  - ✓ La fonction **`ToUpper()`** se rattachant à la chaîne de caractères en question (considérée comme un objet) permet cette conversion. Elle s 'utilise comme suit :
  - **Code : VB.NET**

```
Dim MonString As String = "Phrase de test"
MonString = MonString.ToUpper()
```

Cette phrase sera donc mise en **MAJUSCULES**.

- *Mettre en minuscules*
  - ✓ Cette fonction s 'appelle **`ToLower()`** ; elle effectue la même chose que la précédente, sauf qu'elle permet le formatage du texte en minuscules.
  - **Code : VB.NET**

```
Dim MonString As String = "Phrase de test"
MonString = MonString.ToLower()
```

Jouer avec les mots, les dates

Les dates, le temps

- Il s'agit d'un sujet assez sensible puisque, lorsque nous aborderons les bases de données.
- Pour travailler, nous allons avoir besoin d'une date. Ça vous dirait, la date et l'heure d'aujourd'hui ? Nous allons utiliser l'instruction `Date.Now`, qui nous donne… la date et l'heure d'aujourd'hui, sous la forme suivante :

Code : Console

```
16/06/2009 21:06:33
```

- Nous allons ainsi pouvoir travailler. Entrons cette valeur dans une variable de type… `date`, et amusons-nous !

Récupérer uniquement la date

- La première fonction que je vais vous présenter dans ce chapitre est celle qui convertit une chaîne `date`, comme celle que je viens de vous présenter, mais uniquement dans sa partie « date ».
- Je m'explique : au lieu de « 16/06/2009 21:06:33 » (oui, je sais, il est exactement la même heure qu'il y a deux minutes…), nous obtiendrons « 16/06/2009 ».
- **Code : VB.NET**

```
    ToShortDateString()
```

Jouer avec les mots, les dates

Les dates, le temps

- Cette fonction s 'utilise sur une variable de type `date`. J'ignore si vous vous souvenez de mon petit interlude sur les objets et fonctions, au cours duquel je vous ai expliqué que le point (« . ») servait à affiner la recherche. Nous allons donc utiliser ce point pour lier le type (qui est également un objet dans notre cas) et cette fonction.
- Cette syntaxe que vous avez, je pense, déjà écrite vous-mêmes (`MaVariableDate.ToShortDateString()`), convertit votre date en date sans heure, mais flotte dans les airs… Il faut bien la récupérer, non ? Pour ce faire, affichons-la !

Code : VB.NET

```
Module Module1
    Sub Main()
        'Initialisation des variables
        Dim MaVariableDate As Date = Date.Now
        'Écriture de la forme courte de la date
        Console.WriteLine(MaVariableDate.ToShortDateString)
        'Pause
        Console.Read()
    End Sub
End Module
```

Code : Console

```
16/06/2009
```

Jouer avec les mots, les dates

La date avec les libellés.

✓ Seconde fonction : récupérer la date avec le jour et le mois écrits en toutes lettres.

✓ Donc, pour obtenir cela, notre fonction s 'intitule **`ToLongDateString()`**.

- Le résultat obtenu est Mardi 16 Juin 2009 .

L'heure uniquement

✓ Voici la fonction qui sert à récupérer uniquement l'heure :

Code : VB.NET

```
ToShortTimeString()
```

L'heure avec les secondes

- Même chose qu'au-dessus, sauf que la fonction se nomme :

Code : VB.NET

```
ToLongTimeString()
```

Les Tableaux

Un tableau va servir à stocker plusieurs valeurs ; s'il s'agit seulement d'entrer un nombre à l'intérieur, cela ne sert à rien. Par exemple, dans une boucle qui récupère des valeurs, si on demande dix valeurs, on saisit les valeurs dans un tableau.

- **Les dimensions**
  1. ***Tableau à une dimension***

Représentation d'un tableau à une dimension

✓ Comme vous le voyez, c'est exactement comme sous Excel.

✓ Pour déclarer un tableau de ce type en Visual Basic, c'est très simple : on écrit notre déclaration de variable, d'`Integer` (entier) par exemple, et on place **l'*index du tableau entre parenthèses***. *Voici le code source de l'exemple que je viens de vous montrer :*

Code : VB.NET

```
Dim MonTableau(4) As Integer
```

Les tableaux

- Comme sur le dessin, tu disais ? Pourtant, sur ce dernier, il y a cinq cases. Or, tu n'en as inscrites que quatre. Comment cela se fait-il ?
  - ✓ Oui, sa longueur est de 4. Vous devez savoir qu'*un tableau commence toujours par 0. Et donc, si vous avez compris, un tableau* longueur 4 possède les cases 0, 1, 2, 3 et 4, soit cinq cases, comme sur le dessin.
  - ✓ Le nombre de cases d'un tableau est toujours « **indice + 1 ».**
  - ✓ Réciproquement, l'index de sa dernière case est « **taille - 1 ».**
  - ✓ Souvenez-vous de cela, ce sera utile par la suite.
  - ✓ Comment écrire dans un tableau ?
  - ✓ C'est très simple. Vous avez par exemple votre tableau de cinq cases (dimension 4) ; pour écrire dans la case 0 (soit la première case), on écrit ceci :

Code : VB.NET

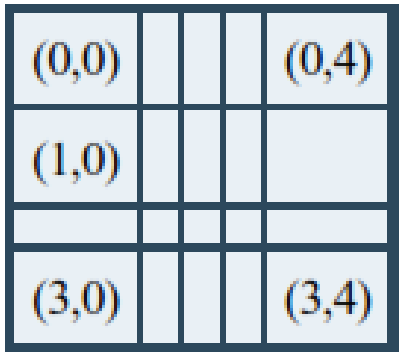
```
MonTableau(0) = 10
```

- Eh oui, il s 'utilise comme une simple variable ! Il suffit juste de mettre la case dans laquelle écrire, accolée à la variable et entre parenthèses.

Les tableaux

2. Tableaux à deux dimensions

- Représentation d'un tableau à deux dimensions
- Il s 'agit ici d'un tableau à **deux dimensions** : une pour la hauteur, une autre pour la largeur. Pour créer ce type de tableau, le code est presque identique :
- **Code : VB.NET**




```
Dim MonTableau(3,4) As Integer
```

- Cela créera un tableau avec quatre lignes et cinq colonnes, comme sur le schéma.
- Pour ce qui est de le remplir, le schéma l'explique déjà très bien :
- **Code : VB.NET**

```
MonTableau(0,0) = 10
```

Cela attribuera « 10 » à la case en haut à gauche.

Les tableaux

3. Tableaux à trois dimensions

- Comme vous le voyez, ce type de tableau est représentable par un « cube ». Il peut être utile lors de représentations tridimensionnelles. Pour le déclarer, je pense que vous avez compris la marche à suivre.

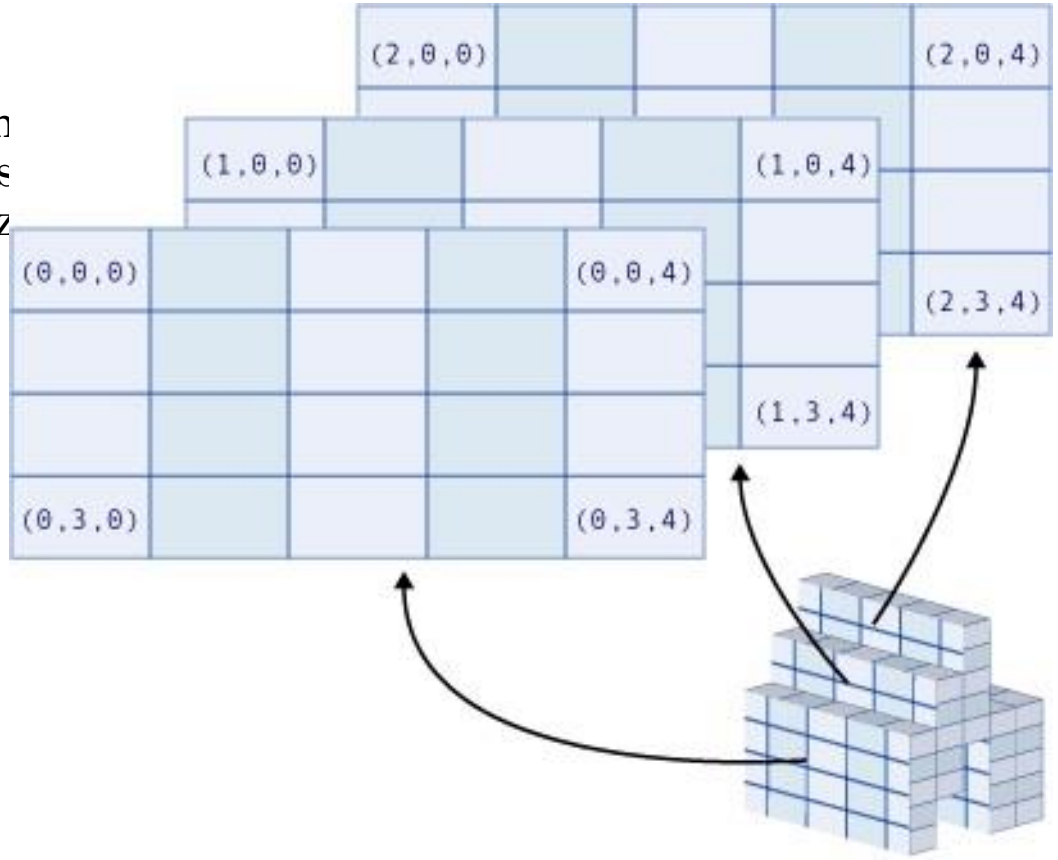
Code : VB.NET

```
Dim MonTableau(2,3,4) As Integer
```

- *Idem pour lui attribuer une valeur :*

Code : VB.NET

```
MonTableau(2,3,4) = 10
```



Les tableaux

4. Autres manipulations avec les tableaux

• *Redimensionner un tableau*

- La taille d'un tableau peut être redimensionnée au cours d'une application. L'instruction **`Redim`** **permet de modifier la taille du** tableau.

Code : VB.NET

```
Redim monTableau(20)
```

- Pour pouvoir conserver le contenu d'un tableau lors d'un redimensionnement, il faut spécifier le mot-clé `Preserve` après **`Redim`**.

Code : VB.NET

```
Redim Preserve monTableau(20)
```

L'instruction **`Redim`** **n'est valable que pour les tableaux à une seule dimension**

• *Retourner un tableau*

- Par exemple, j'ai un tableau qui contient les nombres de 1 à 10, je souhaite avoir ce comptage de 10 à 1. Cette méthode peut alors être utilisée pour effectuer cette opération.

Code : VB.NET

```
Array.Reverse(monTableau)
```

• *Vider un tableau*

Trois paramètres sont nécessaires ici. Le premier est très simplement le tableau à vider, le second spécifie à partir de quel index vider, et le troisième indique le nombre de cases à vider.

```
Array.Clear(monTableau, 0, 10)
```

Les tableaux

4. Autres manipulations avec les tableaux

Copier un tableau dans un autre

- Dernière petite fonction utile, celle permettant de copier un tableau dans un autre.

Code : VB.NET

```
Array.Copy(monTableau1, monTableau2, 5)
```

- Trois paramètres, les deux premiers étant des tableaux. Le premier tableau étant la *source (celui dans lequel nous allons copier les* éléments) et le second est la *destination (celui dans lequel nous allons coller les éléments). Le troisième paramètre est le nombre* d'éléments à copier (depuis l'élément 0). Ainsi, 5 indique que 5 cases seront copiées dans l'autre tableau.
- Si vous souhaitez remplir le second tableau entièrement, utilisez `Array.Copy(monTableau1, monTableau2, monTableau2.Length)`. Cela permet de spécifier que l'on veut copier autant de cases que disponibles dans le deuxième tableau. Nous analyserons ce `.Length` en détail plus tard.

Les fonctions

- Une fonction répète une action bien précise. Nous en connaissons déjà, par exemple **`BEEP`** ou **`IsNumeric()`**, qui vérifie que la valeur d'une variable est bien un nombre.
- Vous voyez, des programmeurs ont déjà créé et intégré des fonctions dans les bibliothèques, d'énormes fichiers qui les rassemblent toutes et que vous possédez sur votre ordinateur dès lors que vous avez installé Visual Basic.
- Nous allons donc à notre tour programmer une fonction et apprendre à l'utiliser.
  - Nous allons donc créer notre première fonction, la plus basique qui soit : sans argument, sans retour. Mais on va tout de même lui faire faire quelque chose… Pourquoi, par exemple, ne pas additionner deux nombres que l'on saisit à l'écran ?
  - Vous vous rappelez certainement le TP avec l'addition. Eh bien, on va factoriser l'addition avec la demande des nombres («factoriser » signifie mettre sous forme de fonction).

Les fonctions

- **Notre première fonction !**

Code : VB.NET

- La ligne `Addition()` appelle donc la fonction, mais pourquoi ? Avez-vous remarqué le **`Sub` que j'ai placé en dessous du** `Main()` ?

```
Module Module1
    Sub Main()
        Addition()
    End Sub
    Sub Addition()
        Dim ValeurEntree As String = ""
        Dim Valeur1 As Integer = 0
        Dim Valeur2 As Integer = 0
        Console.WriteLine("- Addition de deux nombres -")
        'Récupération du premier nombre
        Do
            Console.WriteLine("Entrez la première valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un integer
        Valeur1 = ValeurEntree
        'Récupération du second nombre
        Do
            Console.WriteLine("Entrez la seconde valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un integer
        Valeur2 = ValeurEntree
        'Addition
        Console.WriteLine(Valeur1 & " + " & Valeur2 & " = " &
Valeur1 + Valeur2)
        'Pause factice
        Console.Read()
    End Sub
End Module
```

Les fonctions

- **Ajout d'arguments et de valeur de retour**
  - ✓ ***Les arguments***
    - ✓ ***Exemples:***
      - ✓ Pour la fonction **Write()**, l'argument est la valeur placée entre parenthèses, et cette fonction effectue « Affiche-moi cette valeur ! ».
      - ✓ Pour le **BEEP**, les arguments étaient la fréquence et la durée. Et tous deux séparés par… une virgule (« , ») !

C

```
Sub Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer)
```

✓ Vous remarquez bien les `ByVal Valeur1 As Integer` ; **cette syntaxe est à utiliser pour *chaque argument : le mot* `ByVal`,** le nom de la variable, le mot `As`**, et le type de la variable.**

Les fonctions

- ***Les arguments***
- Ce qui nous donne, dans un cas comme notre addition :

Code : VB.NET

```
Sub Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer)
        'Addition des deux arguments
        Console.WriteLine(Valeur1 & " + " & Valeur2 & " = " & Valeur1 + Valeur2)
        'Pause factice
        Console.Read()
End Sub
```

Voilà par exemple le **`Sub` que j'ai écrit, et qui additionne deux valeurs passées en arguments.**

- Pourquoi n'as-tu pas déclaré les variables `Valeur1` et `Valeur2` ?
- Elles ont été automatiquement déclarées dans la ligne de création de fonction.
- Si vous souhaitez appeler cette fonction, comment faut-il procéder ?

Les fonctions

- ***Les arguments***
- Si vous souhaitez appeler cette fonction, comment faut-il procéder ?

Code : VB.NET

```
Addition(Valeur1, Valeur2)
```

- **Remarques:**
  - Vous n'êtes pas obligés de mettre les mêmes noms du côté de l'appel et du côté de la déclaration des arguments dans votre fonction ; la ligne `Addition(Valeur10, Valeur20)` aurait fonctionné, mais il faut bien sûr que `Valeur10` et `Valeur20` soient déclarées du côté de l'appel de la fonction.
  - Il faut obligatoirement utiliser tous les arguments lors de l'appel de la fonction.
  - Les arguments doivent être passés dans le bon ordre !

Les fonctions

Valeur de retour

- Imaginez que vous ayez envie d'une fonction qui effectue un calcul très compliqué ou qui modifie votre valeur d'une certaine manière. Vous voudriez sans doute récupérer la valeur ? C'est ce qu'on appelle le retour :

Code : VB.NET

```
Function Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer) As Integer
```

- La derniere (**`As Integer`**) **qui nous intéresse : c'est cette partie qui indiquera à la fonction le type de valeur** qu'elle doit renvoyer.

pourquoi as-tu écrit **`Function`** **au début et non plus** **`Sub`** **?**

Les fonctions

- ***Valeur de retour***

✓ les **Sub ne renvoient rien, il faut donc passer par les fonctions si on veut une valeur de retour.**

Code : VB.NET

```
Function Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer) As Integer
        Dim Resultat As Integer
        'Addition des deux arguments
        Resultat = Valeur1 + Valeur2
        'Renvoie le résultat
        Return Resultat
End Function
```

- Cette fonction additionne donc les deux nombres passés en arguments et renvoie le résultat.
- La ligne **Return Resultat est très importante, car c'est elle qui détermine le retour : si vous n'écrivez pas cette ligne, aucun** retour ne se fera.
- Ce qui suit le **Return ne sera pas exécuté ; la fonction est quittée lorsqu'elle rencontre cette instruction. Vérifiez donc** bien l'ordre de déroulement de votre programme.

Les fonctions

- ***Valeur de retour***
  - ✓Comment appeler cette fonction ? La forme `Addition(Valeur1, Valeur2)` aurait pu fonctionner, mais où va la valeur de retour ? Il faut donc récupérer cette valeur avec un « = ».

Code : VB.NET

```
Resultat = Addition(Valeur1, Valeur2)
```

  - ✓Cette fonction peut être directement appelée dans une autre, comme ceci par exemple :

Code : VB.NET

```
Console.WriteLine(Addition(Valeur1, Valeur2))
```

Les fonctions

- **Petits plus sur les fonctions**
  - ***Les arguments facultatifs***
    - ✓ Les arguments facultatifs sont des arguments pour lesquels on peut choisir d'attribuer une valeur ou non au moment de l'appel de la fonction. Pour les déclarer, tapez `Optional ByVal Valeur3 As Integer = 0`.
    - ✓ Le mot-clé `Optional` **est là pour dire qu'il s 'agit d'un argument facultatif, le reste de la syntaxe étant la même que pour les** autres fonctions.
    - ✓ Un argument facultatif doit toujours être initialisé et se faire attribuer une valeur dans la ligne de déclaration de la fonction.
    - ✓ **Code : VB.NET**

```
Function Operation(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer,
Optional ByVal Valeur3 As Integer = 0) As Integer

        Return Valeur1 + Valeur2 + Valeur3
End Function
```

- Dans l'appel de cette fonction, je peux maintenant écrire `Operation(1, 5)`, ce qui me renverra **6, ou alors** `Operation(10, 15, 3)` **qui me renverra 28. Les deux appels sont valides.**

Les fonctions

Commenter une fonction

- Placez-vous sur la ligne juste avant la fonction, ajoutez ensuite trois *quotes : « ''' »*, *d*es lignes vont s 'afficher :

```
''' <summary>
'''
''' </summary>
''' <param name="Valeur1"></param>
''' <param name="Valeur2"></param>
''' <param name="Valeur3"></param>
''' <returns></returns>
''' <remarks></remarks>
```

- Ces lignes permettent de commenter la fonction : dans `summary`, *expliquez brièvement le rôle de la fonction ; dans les* paramètres, précisez à quoi ils correspondent ; et dans la valeur de retour, indiquez ce que la fonction retourne.

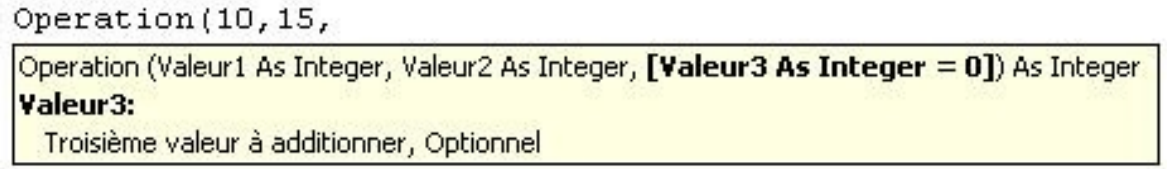
Les fonctions

- Cliquez ensuite sur cette petite flèche pour « replier » cette zone, comme à la figure suivante.

```
''' <summary>
''' Additionne les valeurs passées en arguments
''' </summary>
''' <param name="Valeur1">Permière valeur à additionner</param>
''' <param name="Valeur2">Seconde valeur à additionner</param>
''' <param name="Valeur3">Troisième valeur à additionner, Optionnel</param>
''' <returns>L'addition des arguments</returns>
''' <remarks></remarks>
Function Operation(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer, Optional ByVal Valeur3 As Integer = 0) As Integer
    Return Valeur1 + Valeur2 + Valeur3
End Function
```

Cliquez sur la flèche pour « replier » la zone

- À présent, à chaque endroit où vous allez écrire cette fonction, un cadre s 'affichera vous indiquant ce qu'il faut lui donner comme arguments, comme à la figure suivante.



Un cadre s 'affiche

- Cela est *très utile lorsque vous avez beaucoup de fonctions.*

Boucles supplémentaires

For Each*, *traduisible par « pour chaque »

- Vous vous souvenez des tableaux ?

Code : VB.NET

```
Dim MonTableau(9) As integer
```

- La boucle **`For Each`** **permet de parcourir ce tableau (un peu à la manière du `For` traditionnel) et de récupérer les** valeurs.
- Utilisons donc immédiatement cette boucle :

Code : VB.NET

```
For Each ValeurDeMonTableau As Integer In MonTableau
    If ValeurDeMonTableau < 10 Then
        Console.WriteLine(ValeurDeMonTableau)
    End If
Next
```

Boucles supplémentaires

- ***For Each*, traduisible par « pour chaque »**
  - ✓ Ce qui se traduit en français par ceci : « Pour chaque `ValeurDeMonTableau` qui sont des entiers dans `MonTableau` ».
  - ✓ Ce code parcourt mon tableau et vérifie si chaque valeur est inférieure à 10 ; si c'est le cas, il l'affiche.
  - ✓ On ne peut pas assigner de valeur dans un **`For Each`, seulement les récupérer.**
  - ✓ Très utile, donc, pour lire toutes les valeurs d'un tableau, d'un objet liste par exemple (que nous verrons plus tard).

- Un **`For Each` peut être utilisé pour parcourir chaque lettre d'un mot :**

Code : VB.NET

```
Dim MaChaine As String = "Salut"
Dim Compteur As Integer = 0
For Each Caractere As String In MaChaine
    If Caractere = "a" Then
        Compteur = Compteur + 1
    End If
Next
```

- Ce code compte le nombre d'occurrences de la lettre *a dans un mot.*

Boucles supplémentaires

- ***IIF***
  - **`IIF`** est très spécial et peu utilisé : en un certain sens, il simplifie le **`If`**. **Voici un exemple de son utilisation dans le code** précédent :

Code : VB.NET

```
Dim MaChaine As String = "Salut"
Dim Compteur As Integer = 0
For Each Caractere As String In MaChaine
    If Caractere = "a" Then
        Compteur = Compteur + 1
    End If
Next
Console.WriteLine(IIF(Compteur > 0, "La lettre 'a' a été trouvée
dans " & MaChaine, "La lettre 'a' n'a pas été trouvée dans " &
MaChaine))
```

- si vous avez bien analysé : si le premier argument est vrai (c'est un booléen), on retourne le second argument ; à l'inverse, s 'il est faux, on retourne le dernier. Pour mieux comprendre :

Code : VB.NET

```
IIF(MonBooleen, "MonBooleen est true", "MonBooleen est false")
```

- MonBooleen peut bien évidemment être une condition.

Boucles supplémentaires

Les casts

- un *cast convertit une variable d'un certain type en un autre.* Il existe plusieurs moyens d'effectuer des *casts : une fonction universelle, et d'autres plus spécifiques.*

❖ `Ctype()`

- La fonction universelle se nomme **`Ctype`. Voici sa syntaxe :**

Code : VB.NET

```
Ctype(MaVariableString, Integer)
```

- Ce code convertira `MaVariableString` en `integer`.
- Voici un exemple concret :

Code : VB.NET

```
Dim MonString As String = "666"
If Ctype(MonString, Integer) = 666 Then
'...
End If
```

Les fonctions spécifiques

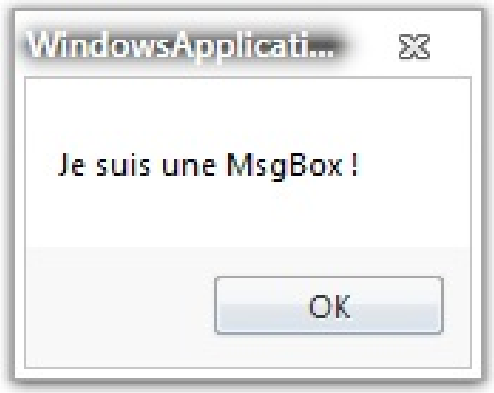
- On a vu l'exemple de **`Ctype()`**, **utile lorsqu'il s 'agit de types peu courants. Mais pour les types courants, il existe des fonctions** plus rapides et adaptées :

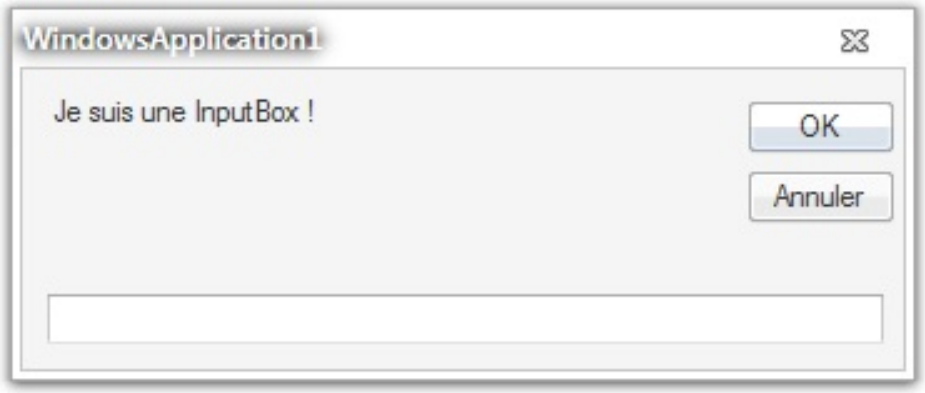
  - ✓ **`CBool()` : retourne un `Boolean`.**
  - ✓ **`CByte()` : retourne un `Byte`.**
  - ✓ **`CChar()` : retourne un `Char`.**
  - ✓ ***`CDate()` : retourne une date.**
  - ✓ ***`CDbl()` : retourne un `Double`.**
  - ✓ **`CDec()` : retourne un nombre décimal.**
  - ✓ ***`CInt()` : retourne un `Integer`.**
  - ✓ **`CLng()` : retourne un `Long`.**
  - ✓ **`CSng()` : retourne un `Single`.**
  - ✓ ***`CStr()` : retourne un `String`.**
  - ✓ **`CUInt()` : retourne un `Unsigned Integer`.**
  - ✓ **`CULng()` : retourne un `Unsigned Long`.**
  - ✓ **`CUShort()` : retourne un `Unsigned Short`.**

  « * » sont les plus utilisées

Les MsgBox et InputBox

- Deux petites choses qui peuvent également vous aider : les **`MsgBox` et les `InputBox` (voir figures suivantes).**





- Les `MsgBox` peuvent signaler une erreur, demander une confirmation, etc. Les `InputBox`, quant à elles, peuvent être utilisées dans des scores par exemple, pour demander le nom du joueur.

Les MsgBox et InputBox

- **La MsgBox**
  - *Les paramètres*

- Voici la liste des arguments. Pas de panique, il n'y en a que trois ! Je vais vous les décrire :
  1. `Prompt` : message qui apparaîtra dans la `MsgBox`.
  2. `Buttons` : type de bouton à utiliser (style de la boîte).
  3. `Title` : titre de la boîte.

  Divers exemples de style

Membre	Valeur	Description
`OKOnly`	0	Affiche le bouton « OK » uniquement.
`OKCancel`	1	Affiche les boutons « OK » et « Annuler ».
`AbortRetryIgnore`	2	Affiche les boutons « Abandonner », « Réessayer » et « Ignorer ».
`YesNoCancel`	3	Affiche les boutons « Oui », « Non » et « Annuler ».
`YesNo`	4	Affiche les boutons « Oui » et « Non ».
`RetryCancel`	5	Affiche les boutons « Réessayer » et « Annuler ».
`Critical`	16	Affiche l'icône « Message critique ».
`Question`	32	Affiche l'icône « Requête d'avertissement ».
`Exclamation`	48	Affiche l'icône « Message d'avertissement ».
`Information`	64	Affiche l'icône « Message d'information ».
`DefaultButton1`	0	Le premier bouton est le bouton par défaut.
`DefaultButton2`	256	Le deuxième bouton est le bouton par défaut.
`DefaultButton3`	512	Le troisième bouton est le bouton par défaut.
`ApplicationModal`	0	L'application est modale. L'utilisateur doit répondre au message avant de poursuivre le travail dans l'application en cours.
`SystemModal`	4096	Le système est modal. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message.
`MsgBoxSetForeground`	65536	Spécifie la fenêtre de message comme fenêtre de premier plan.

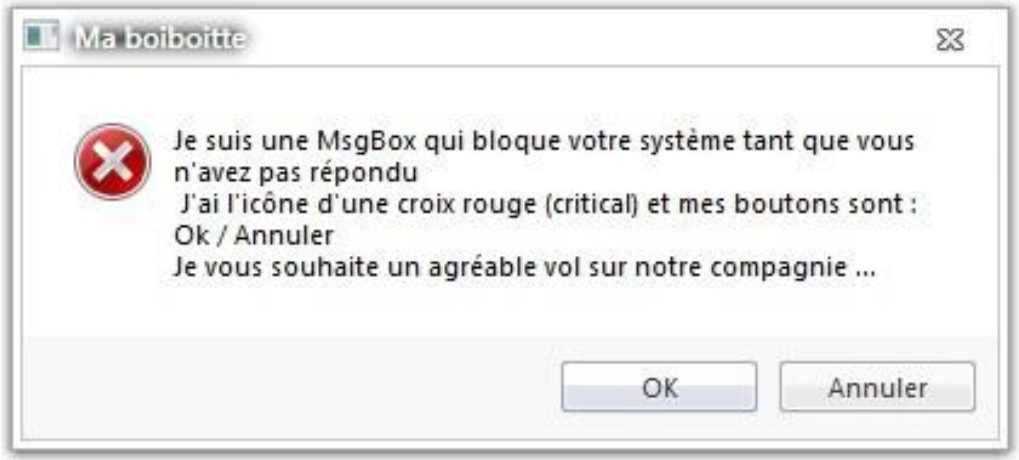
Les MsgBox et InputBox

- **La MsgBox**
  - ***Les paramètres***
    - Les numéros indiqués correspondent aux *ID, que vous pouvez cumuler. En gros, si vous souhaitez que votre boîte bloque le* système et que l'on doive y répondre avant de continuer, avec une icône « Message critique » et des boutons « OK - Annuler », il faut que vous tapiez… 4096 + 1 + 16 = 4113 !
    - Voici donc le code correspondant, les **`Chr(13)` représentant des retours à la ligne :**

Code : VB.NET

```
MsgBox("Je suis une MsgBox qui bloque
votre système tant que vous
n'avez pas répondu" & Chr(13) & " J'ai
l'icône d'une croix rouge
(critical) et mes boutons sont : Ok /
Annuler" & Chr(13) & "Je vous
souhaite un agréable vol sur notre
compagnie ...", 4113, "Ma
boiboitte")
```

Cela donne le résultat visible à la figure suivante.



Les MsgBox et InputBox

- La `MsgBox`
  - *Le retour*
    - Passons à la valeur de retour !
    - Les boutons sur lesquels on clique ne renvoient pas tous la même valeur :

    1. OK → 1
    2. Cancel → 2
    3. Abort → 3
    4. Retry → 4
    5. Ignore → 5
    6. Yes → 6
    7. No → 7

Un petit `If` **devrait vous permettre d'effectuer une action précise en fonction de ce que l'utilisateur a entré !**

- **`InputBox`**

  ✓ ***Les paramètres***

  Les arguments de l'`InputBox` sont un peu moins ennuyeux et ne sont pas difficiles à retenir :

  1. Le `Prompt`, comme pour la `MsgBox`.
  2. Le titre de la fenêtre.
  3. La valeur par défaut entrée dans le champ à remplir.
  4. La position de la boîte en X (sur l'horizontale, en partant de la gauche).
  5. La position de la boîte en Y (sur la verticale, en partant du haut).

  L'origine du repère se situe donc en haut à gauche. Si vous entrez « 0 » pour les positions X et Y, alors la boîte se retrouvera en haut à gauche ; pour la laisser centrée, ne mettez rien.

- **InputBox**

  ✓ ***Le retour***

  Cette fois, la valeur de retour n'est pas forcément un nombre : cela peut être une chaîne de caractères ou toute autre chose que l'utilisateur a envie d'écrire.

  Voilà pour les *box, c'est fini !*

  - Les constantes sont des variables destinées à ne pas changer de valeur.
  - **`For each` permet d'itérer sur chaque valeur d'une liste, d'un tableau.**
  - `MsgBox` et `InputBox` sont des fenêtres de dialogue pour capter l'attention de l'utilisateur.