Excel
Programmation VBA
Visual Basic pour Applications

Hervé Hocquard

http://www.labri.fr/perso/hocquard

Programmation sous Excel via VBA (Visual Basic pour Applications)

Fonctions personnalisées
Complètement standardisée. Valable pour les autres classeurs et même, si pas d'accès aux objets spécifiques d'Excel, pour les autres outils Office.

Macros
Manipulation directe des objets Excel (classeurs, feuilles, cellules, graphiques, etc.)

Programmation de tâches
Programmation de séquences d'actions plus ou moins complexes, faisant intervenir ou non des mécanismes algorithmiques.

Interfaces graphiques
Boîtes de dialogues standards. Mais aussi les formulaires personnalisées pour faciliter les accès aux fonctionnalités. Nécessite une certaine formalisation et la connaissance des principes de la programmation évènementielle.

Enregistreur de macros
Transformation de séquences d'action en programme VBA. Ne nécessite pas la connaissance de la programmation, mais peu de souplesse (structure fixe, peu adaptable…)

Programmation des macros
Très puissant. Nécessite la connaissance des principes de la programmation et de la syntaxe d'accès aux objets Excel.

Points importants. Connaissance de l'algorithmie, langage de programmation Visual Basic. Les instructions sont écrites dans des fonctions (*function*) et procédures (*sub*), qui sont regroupées dans des modules. Nous travaillons dans VBE (Visual Basic Editor).

Généralités sur la programmation

ALGORITHMIE - PROGRAMMATION

Algorithmie

• Solution « informatique » relative à un problème

• Suite d'actions (instructions) appliquées sur des données

• 3 étapes principales :

1. saisie (réception) des données
2. Traitements
3. restitution (application) des résultats

Programme

• Transcription d'un algorithme avec une syntaxe prédéfinie

• **Visual Basic pour Applications**

• Même principes fondamentaux que les autres langages objets (Java, C#, etc.)

• VBA agit en interaction avec les fonctions prédéfinies disponibles dans la suite Office

Langage interprété : + portabilité application ; - lenteur (R, **VBA**, Python…)

Langage compilé : + rapidité ; - pas portable

(solution possible : write once, compile anywhere ; ex. Lazarus)

Langage pseudo-compilé : + portabilité plate-forme ; - lenteur (?)

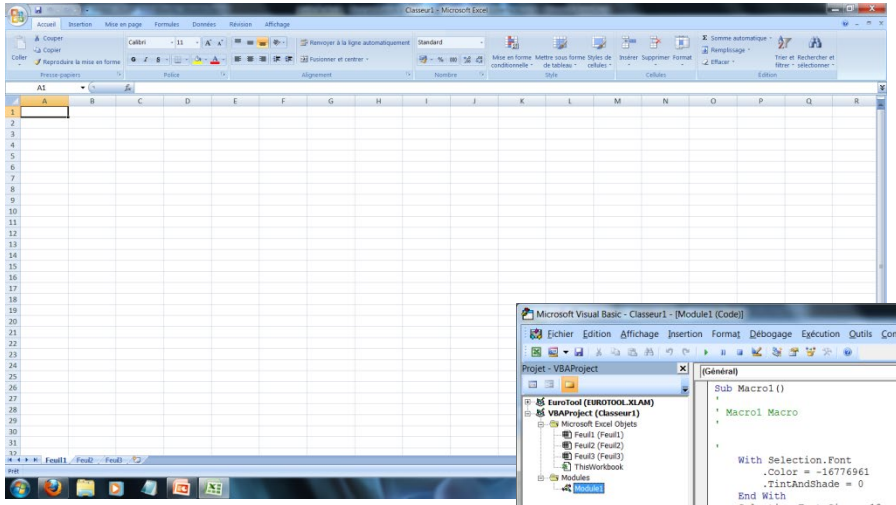
(principe : write once, run anywhere ; ex. Java et le principe JIT)

VBA (Visual Basic pour Applications) est un langage de programmation dédié principalement aux applications Microsoft Office. Il est basé sur le langage Visual Basic, mais ne peut s'exécuter que dans une application hôte Microsoft Office, et non de manière autonome.

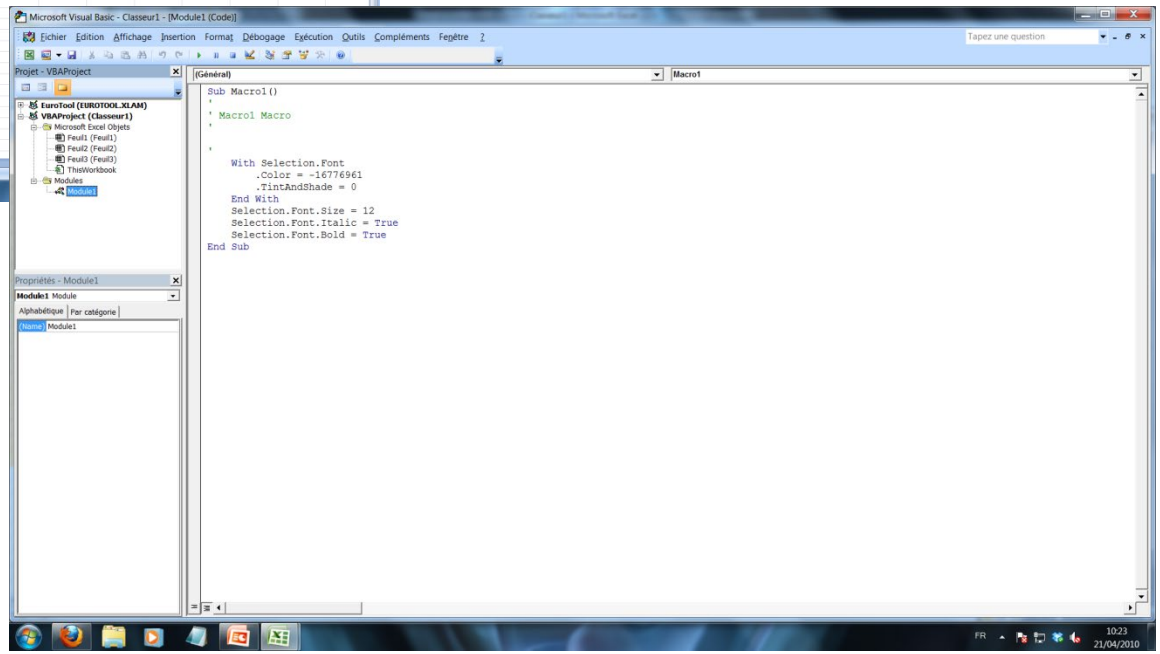
1. Déterminer les besoins et fixer les objectifs : que doit faire le logiciel, dans quel cadre va-t-il servir, quels seront les utilisateurs types ? On rédige un cahier des charges avec le commanditaire du logiciel (Remarque : commanditaire = **maître d'ouvrage** ; réalisateur = **maître d'œuvre**)
2. Conception et spécifications : quels sont les fonctionnalités du logiciel, avec quelle interface ?
3. Programmation : modélisation et codage
4. Tests : obtient-on les résultats attendus, les calculs sont corrects, y a-t-il plantage et dans quelles circonstances ? (tests unitaires, tests d'intégration, etc.)
5. Déploiement : installer-le chez le client (vérification des configurations, installation de l'exécutable et des fichiers annexes, etc.)
6. Maintenance : corrective, traquer les bugs et les corriger (patches) ; évolutive (ajouter des fonctionnalités nouvelles au logiciel : soit sur l'ergonomie, soit en ajoutant de nouvelles procédures)

L'ÉDITEUR VBE ET LE MODÈLE OBJET VBA

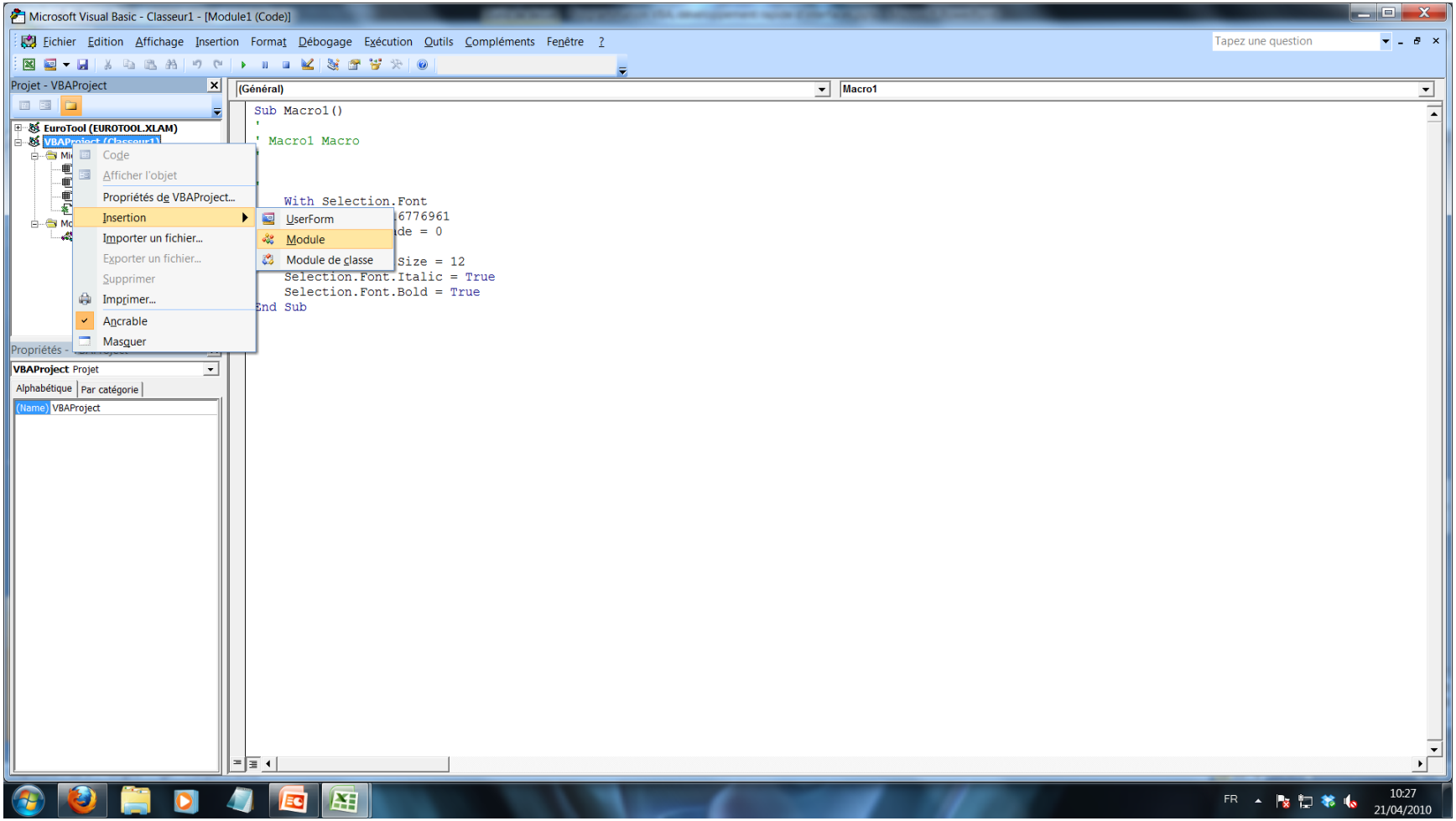
L'éditeur (Visual Basic Editor)



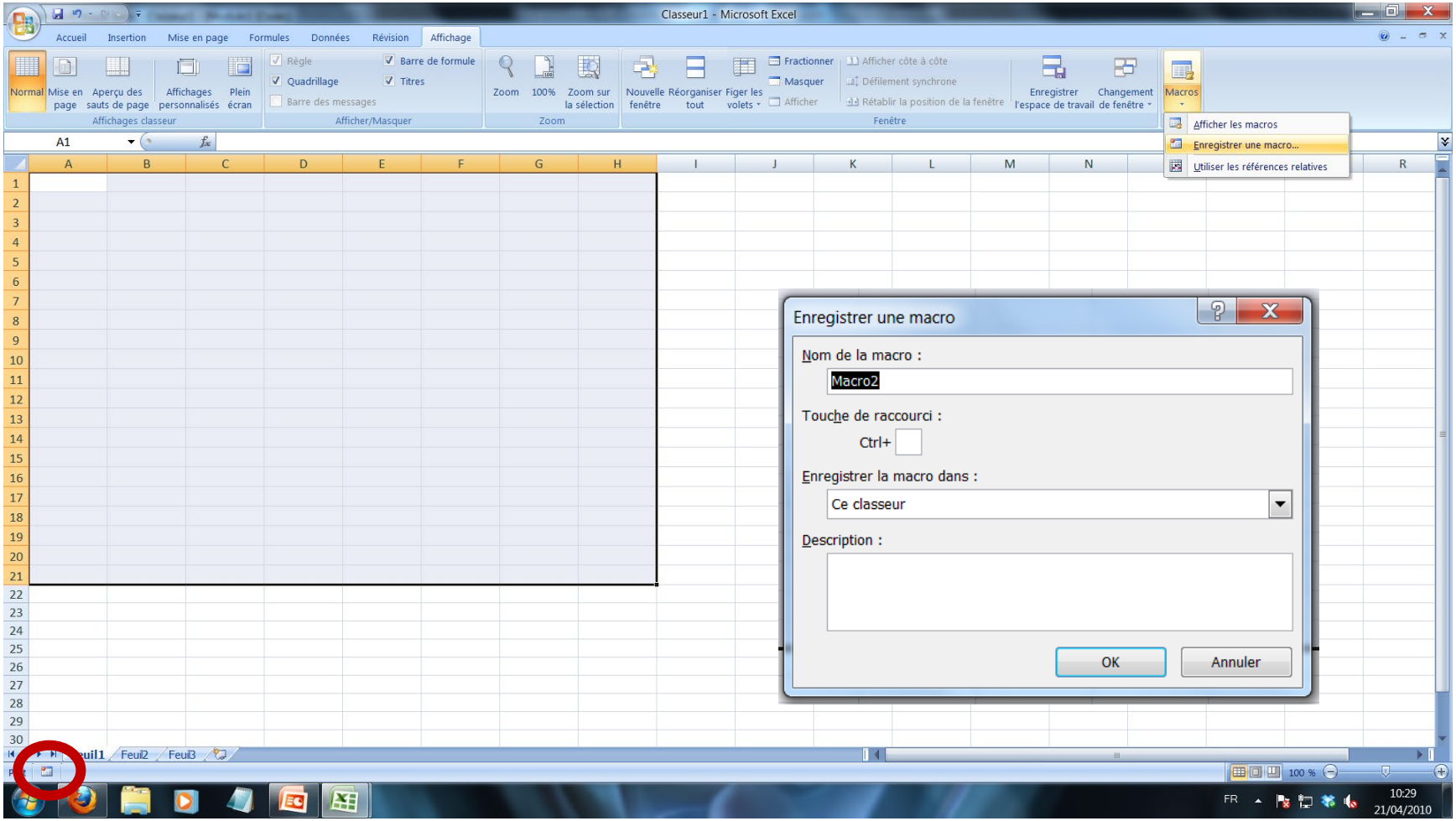
Alt + F11



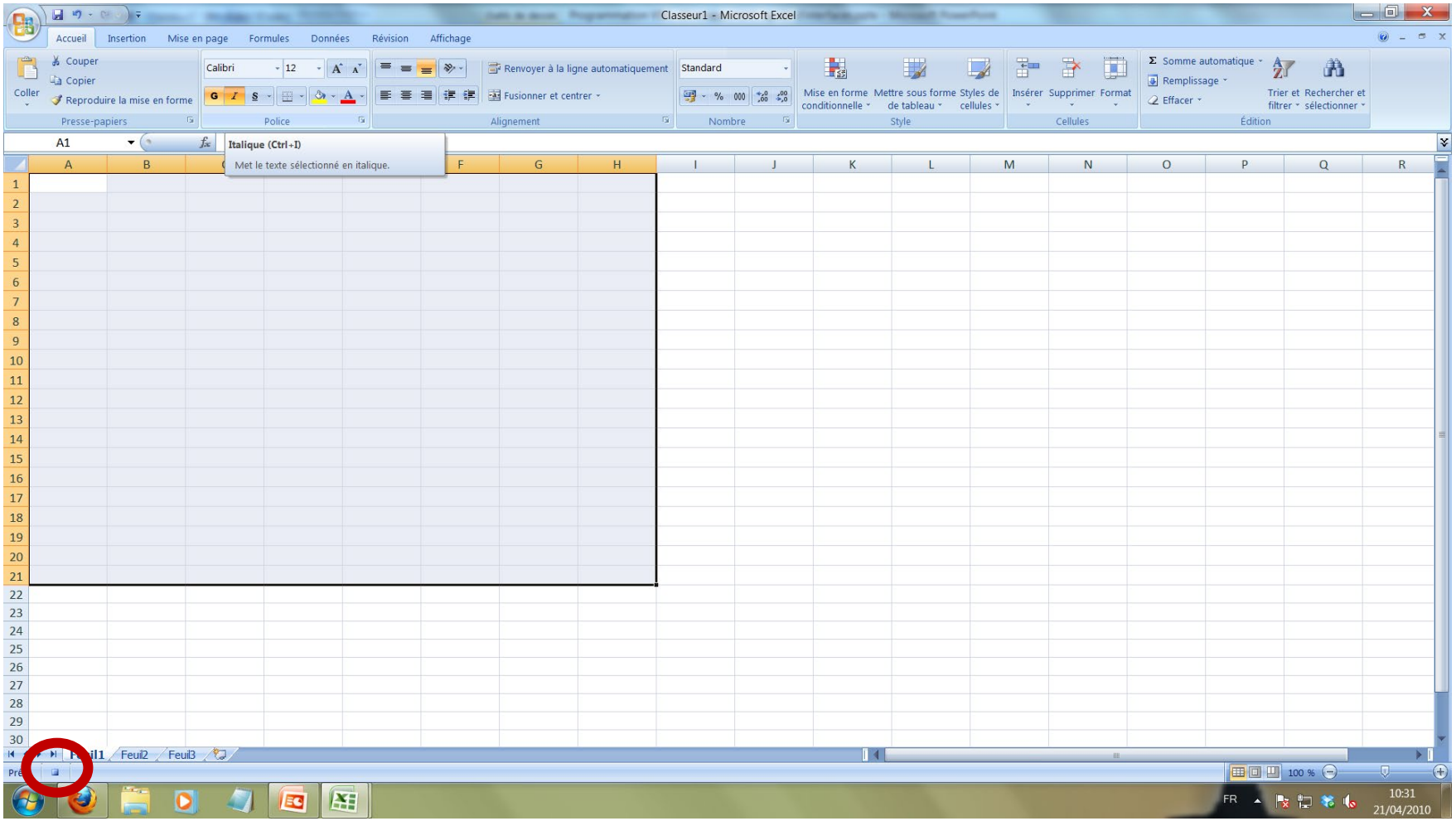
Insertion d'un module dans l'éditeur



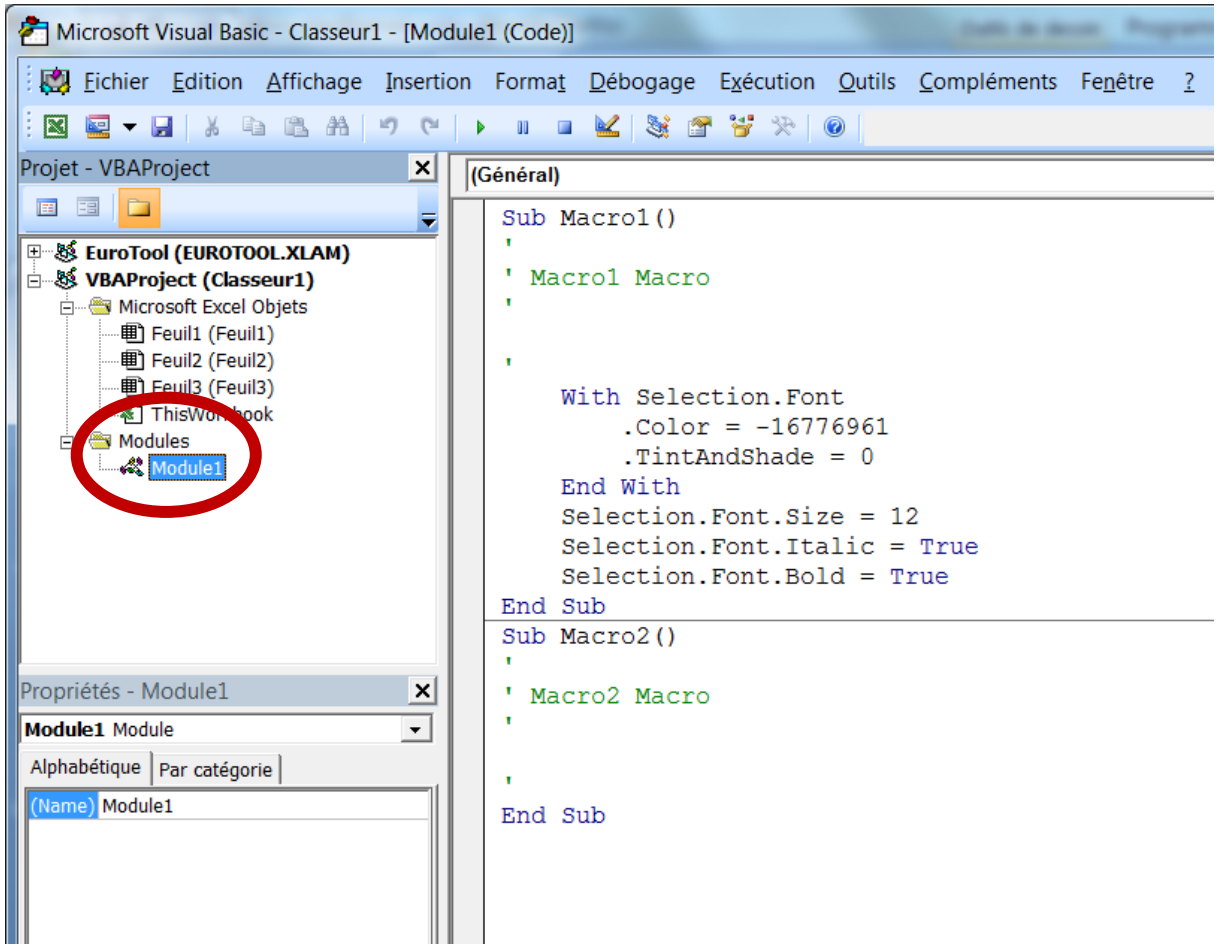
Enregistrer une macro



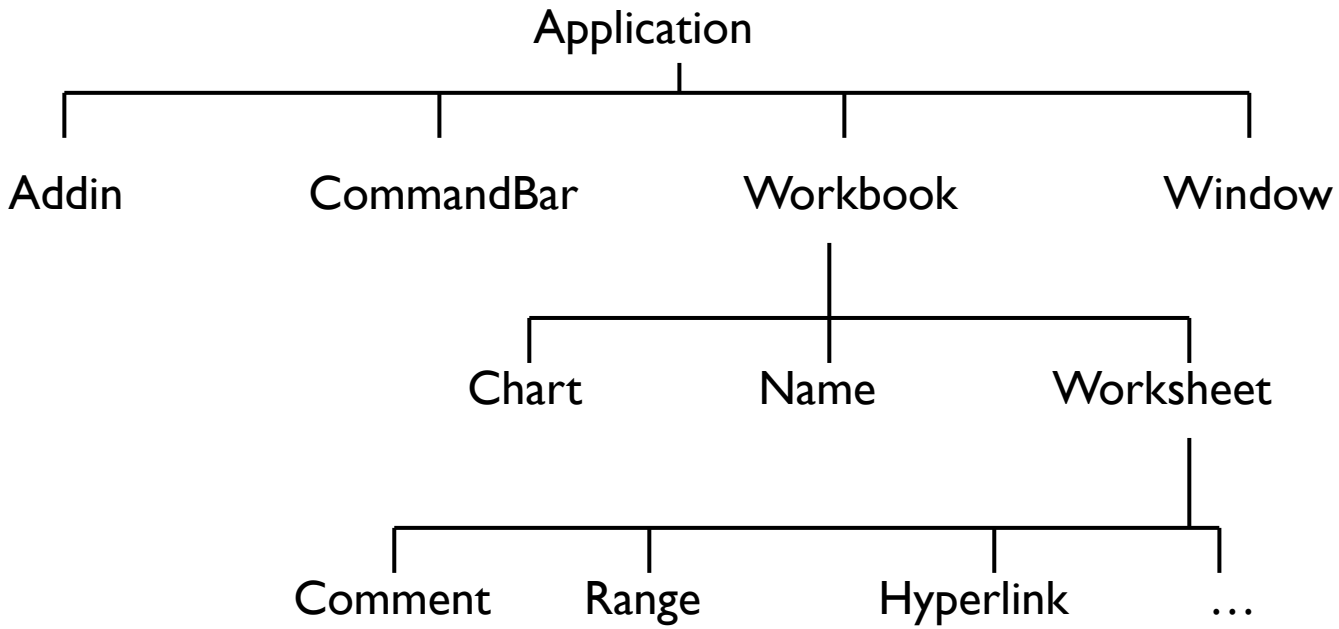
Enregistrer une macro



Impact dans l'éditeur



- Un objet est constitué d'attributs (ou propriétés) et de méthodes qui lui sont associées
- Les objets existants sont constitués en hiérarchie (relation de composition)

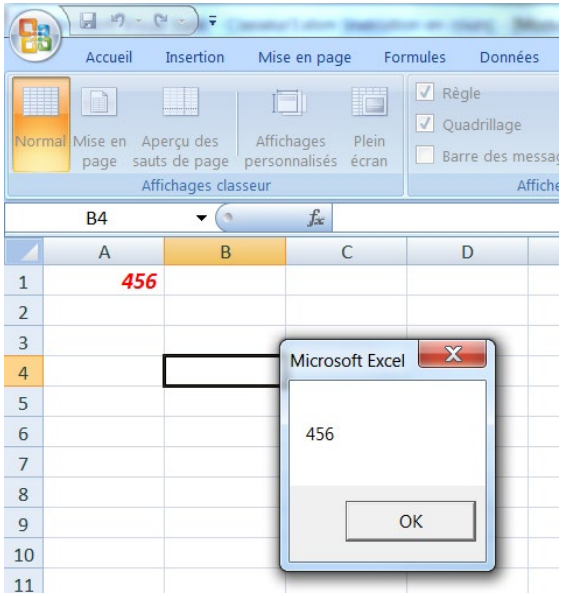


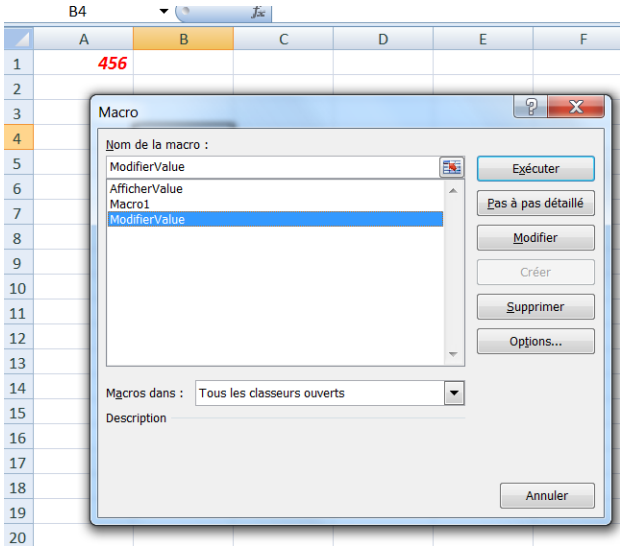
- Concept clé
- On rajoute un « s »!
  - Workbooks : collection des objets Workbook
  - Worksheets : collection des objets Worksheet
  - … etc.

- Faire appel à un élément d'une collection: 2 méthodes:
  - Appel par le nom de l'élément
    - Ex: Worksheets("Feuil1")
  - Appel par l'indice
    - Ex: Worksheets(1)

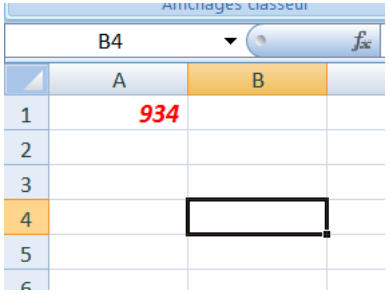
- **Opérateur point (.)**
  - Exemple:
    Application.Workbooks("Classeur1.xlsx").Worksheets(1).Range("A1").Value=9
- **Simplification: par exemple si Classeur1.xlsx est le classeur actif:**
  - Worksheets(1).Range("A1").**Value**=9

```
Sub AfficherValue()
'
Contents = Worksheets("Feuil1").Range("A1").Value
MsgBox Contents


    ActiveWorkbook.Save
End Sub
```






```
Sub ModifierValue()
    Worksheets("Feuil1").Range("A1").Value = 934
End Sub
```



- **Action relative à un objet**

- **Exemples:**
  - Worksheets("Feuil1").Activate
  - Range("A1").Copy Range("B1")

- **Une méthode prend en compte 0, 1 ou plusieurs arguments.**
  - Le premier argument est séparé de la méthode par un espace, les arguments sont séparés entre eux par des virgules
  - OU utilisation des parenthèses

Programme : suite d'instructions manipulant des données

LANGAGE VISUAL BASIC

Données typées. Visual Basic propose les types usuels de la programmation : entier, réels, booléens, chaîne de caractères.

Structures avancées de données. Gestion des collections de valeurs (énumérations, tableaux) et des objets structurés (enregistrements, classes).

Séquences d'instructions, c'est la base même de la programmation, pouvoir écrire et exécuter une série de commandes sans avoir à intervenir entre les instructions.

Structures algorithmiques : les branchements conditionnels et les boucles.

Les outils de la programmation structurée : pouvoir regrouper du code dans des procédures et des fonctions. Organisation du code en modules et possibilité de distribuer ces dernières.

Visual Basic n'est pas « case sensitive », il ne différencie pas les termes écrits en minuscule et majuscule.

Le type de données définit le type d'opérateurs qu'on peut leur appliquer.

- **Numérique** qui peut être réel (double) ou entier (long). Les opérateurs applicables sont : +, -, *, / (division réelle), \ (division entière), mod (modulo)

Exemple : 5 / 2 → 2.5 ; 5 \ 2 → 2 ; 5 mod 2 → 1

- **Booléen** (boolean) qui ne prend que deux valeurs possibles : True et False. Les opérateurs sont : not, and, or.

Exemple : True and False → False

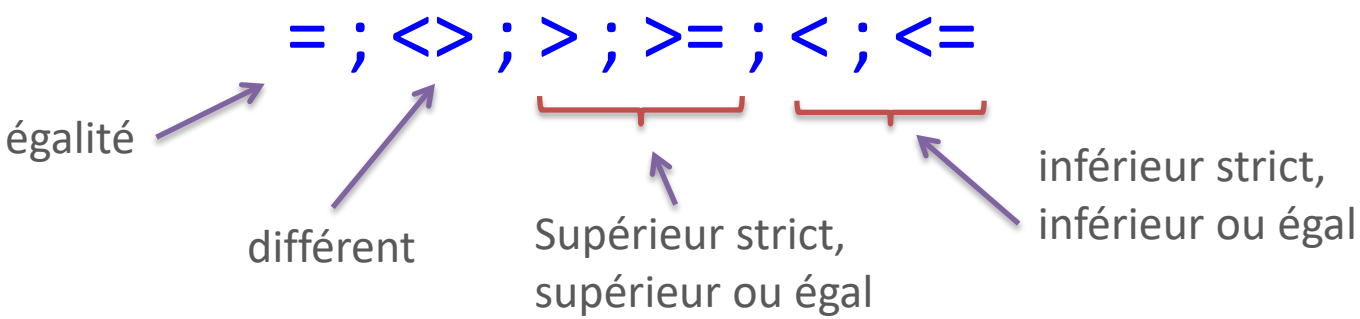
- **Chaîne de caractères** (string) qui correspond à une suite de caractères délimitée par des guillemets '' ''. Les opérateurs possibles sont la concaténation, la suppression d'une sous-partie, la copie d'une sous-partie, etc.

Exemple : ''toto'' est une chaîne de caractères, toto on ne sait pas ce que c'est (pour l'instant)

Habituellement, les opérations font intervenir des données de type identique et renvoie un résultat du même type.

Type	Valeurs
Boolean	Vrai, faux
Integer	Entiers
Long	Entiers
Single	Réels
Double	Réels
Currency	4 chiffres après la ,
Date	1/1/100 à 31/12/9999
String	Chaines de caractères
Object	Tout objet
Variant	N'importe quel type

Les opérateurs de comparaison confrontent des données de même type, mais le résultat est un booléen



Exemples

5 > 2 → True

5 > ''toto'' → illicite

5 <> 5 → False

''toto'' > ''tata'' → True

Licite. Comparaison de gauche à droite basée sur le code ASCII. Arrêt des comparaisons dès que l'indécision est levée.

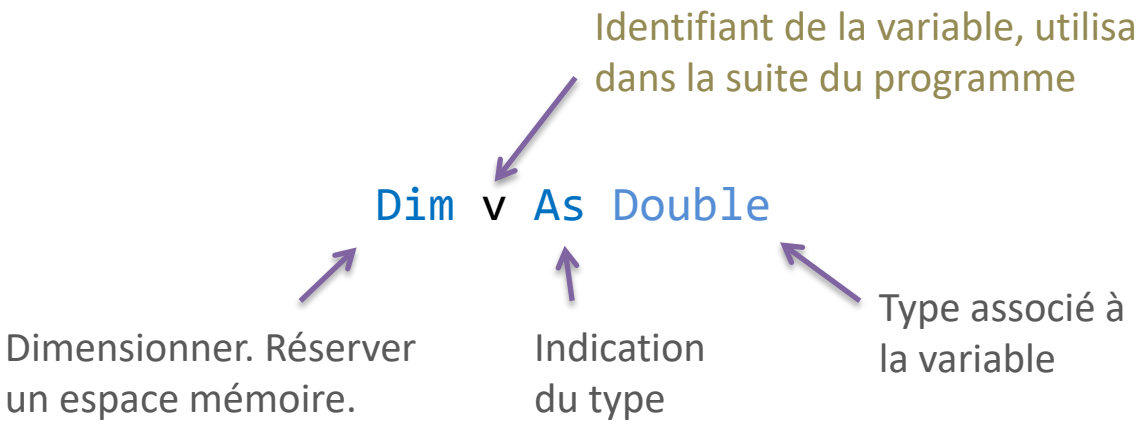
- Valeur absolue: Abs(-9) retourne 9
- Signe: Sgn(-18) retourne –1 (ou 0 ou 1)
- Troncature à l'unité : Fix(-18.3) = -18
  Fix(18.3) = 18
  - Tronque la partie décimale
- Partie entière: Int(13.12) retourne 13
  Int(-14.8) retourne –15
  - $E(x) \leq x < E(x) + 1$
  - Tronque à l'entier inférieur le plus proche.

- Sqr, Exp, Log
  - Sqr(4) retourne 2, Exp(5) retourne 148.413…,
    Log(9) retourne 2.197224… (en base e)
- Nombres aléatoires
  - Rnd retourne un nombre aléatoire entre 0 (compris) et 1 (non compris)
  - a = Rnd a peut valoir 0.12131441
  - Int((b – a + 1) * Rnd + a) retourne un nombre aléatoire entier entre a et b
- Sin, Cos, Tan, Atn (arc-tangente)

- Date retourne la date actuelle
- Time retourne l'heure courante
  - Date et Time peuvent retourner des chaînes de caractères String
- DateSerial retourne une valeur unique pour une date donnée, sous forme Variant
  - dv1 = DateSerial(2003, 4, 22)
    dv2 = DateSerial(1928, 5, 3)
    dv1 – dv2 représente le nombre de jours entre ces deux dates
- Day, Month et Year retourne respectivement le jour, le mois et l'année d'une date.
  - Year(Date) retourne 2019 cette année (en entier)

Les **variables** correspondent à des identifiants auxquels sont associés des valeurs d'un type donné. Elles matérialisent un espace mémoire avec un contenu que l'on peut lire ou écrire.

Déclaration d'une variable



Affectation. Attribuer une valeur à la variable

```
v = 2.5
```

= est le symbole d'affectation. A gauche de = on modifie le contenu dans une variable, à droite on lit le contenu d'une variable. C'est pour cette raison que l'instruction v = v + 1 est licite.

Opération et affectation

```
x = v * 2
```

La valeur 5 est écrite dans la variable x qui doit être déclarée au préalable.

Ecriture et utilisation des fonctions personnalisées dans Excel

FONCTIONS PERSONNALISÉES

Une fonction personnalisée est une fonction VBA qui peut être appelée dans un classeur Excel. Elle prend en entrée des informations en provenance des feuilles du classeur (principalement) et renvoie une valeur insérée dans une cellule (le plus souvent également).

Formalisme

```
Function NomFonction(paramètres) As type de donnée
```

Est un identifiant qui doit respecter la syntaxe VBA

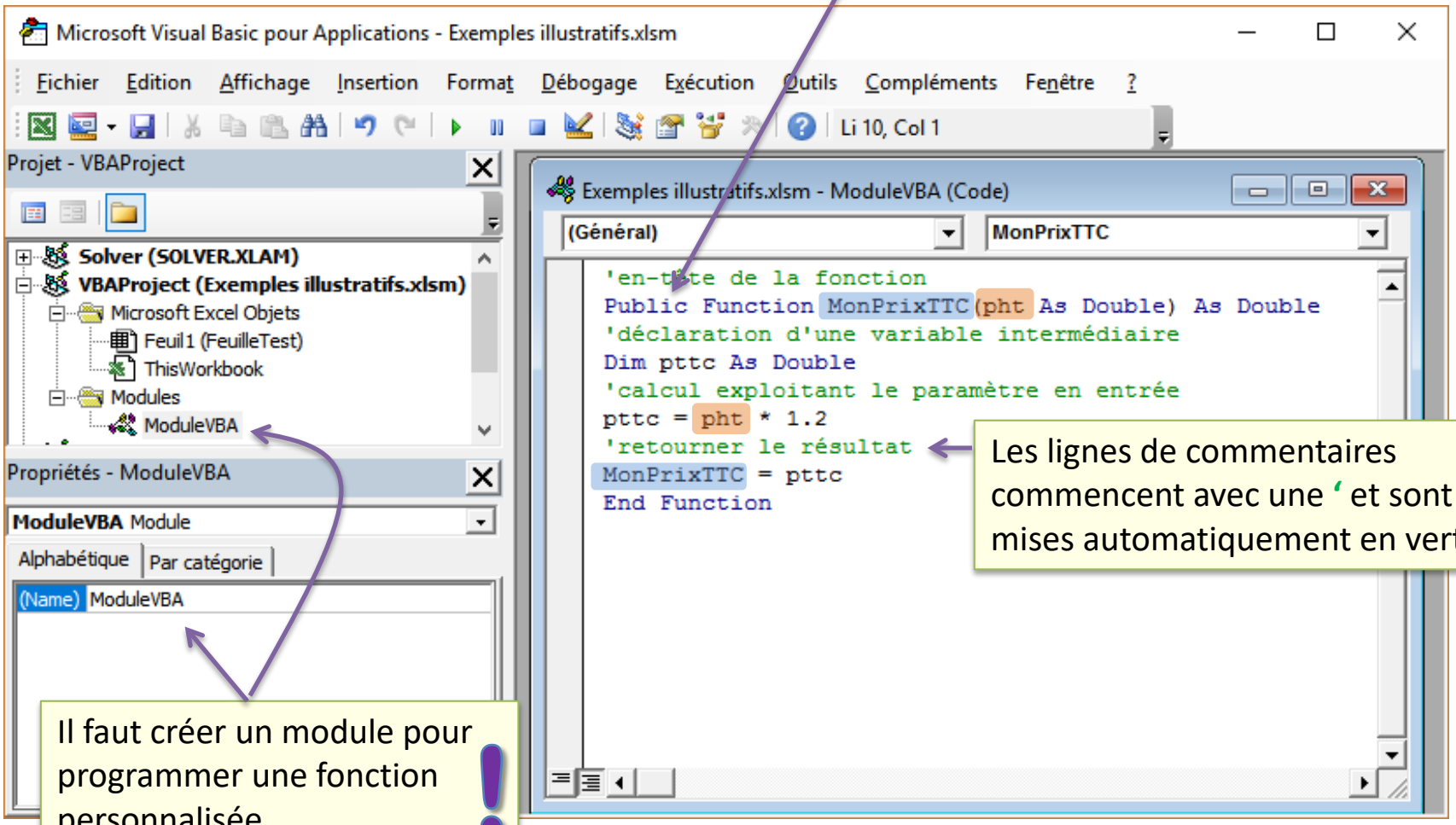
Type de la valeur retournée par la fonction.

Les informations que prend en entrée la fonction, elles prennent la forme *nom_parametre as type de paramètre*. Il peut y en avoir plusieurs, ils sont séparés par des « , » dans ce cas.

Un classeur Excel contenant du code VBA doit être enregistré au format XLSM, prenant en charge les macros. Sinon on perd son code.

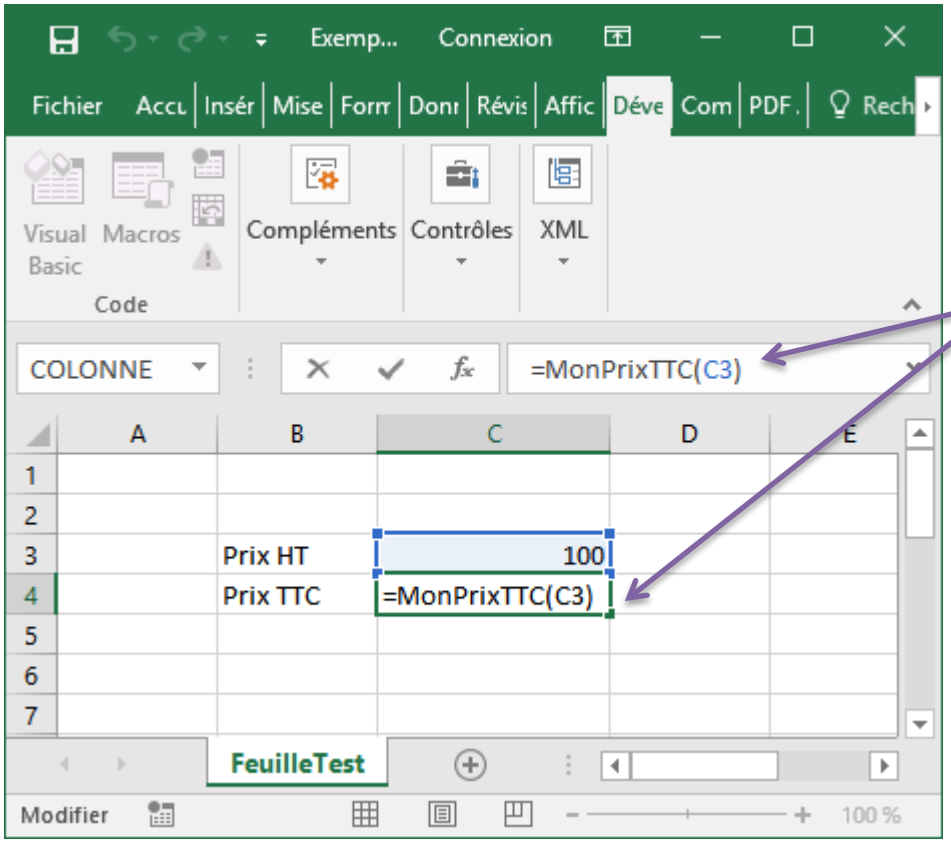
Entrée : prix HT (réel)
Sortie : prix TTC (réel)

Public pour que la fonction soit visible en dehors du module, notamment dans la feuille de calcul

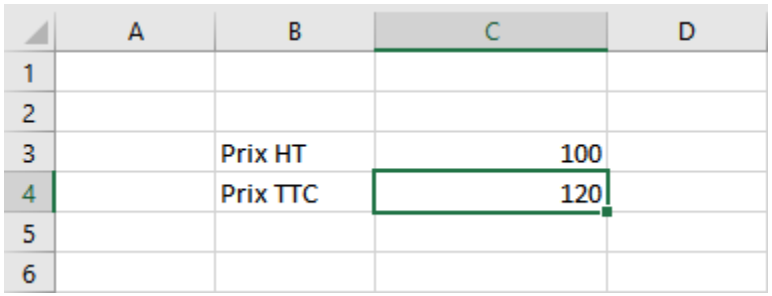


Les lignes de commentaires commencent avec une ' et sont mises automatiquement en vert.

Il faut créer un module pour programmer une fonction personnalisée



La fonction est insérable dans la feuille de calcul comme n'importe quelle autre fonction Excel. Elle est accessible dans la catégorie « Fonctions personnalisées ».

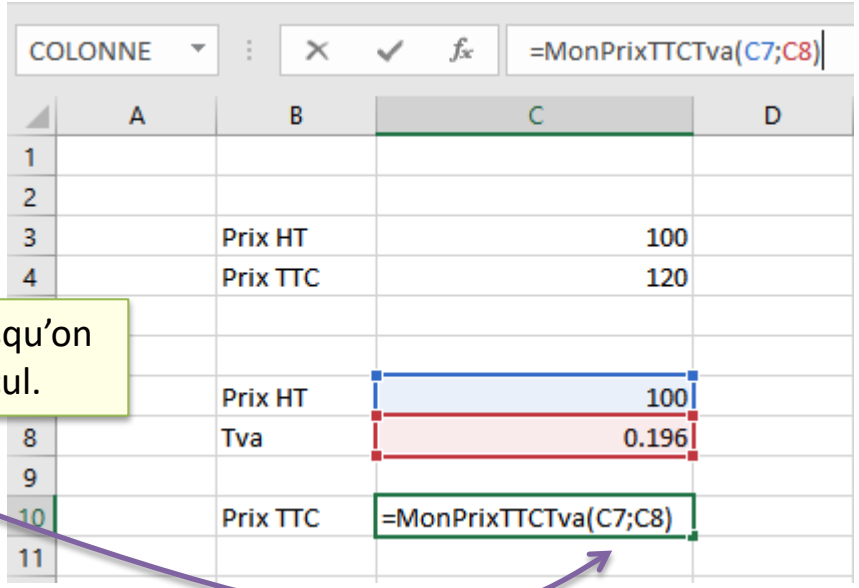


Le résultat s'affiche une fois la fonction insérée et validée. La fonction est automatiquement appelée à chaque fois que la feuille a besoin d'être recalculée (comme pour les autres fonctions standards d'Excel).

Entrées : prix HT (réel), tva (réel)
Sortie : prix TTC (réel)

(1) Le séparateur de paramètres est la « , » lors de la définition de la fonction.

```
'fonction avec 2 paramètres
Public Function MonPrixTTCTva(pht As Double, tva As Double) As Double
'déclaration d'une variable intermédiaire
Dim pttc As Double
'calcul exploitant les paramètres en entrée
pttc = pht * (1 + tva)
'retourner le résultat
MonPrixTTCTva = pttc
End Function
```



(2) Très étrangement, il devient « ; » lorsqu'on appelle la fonction dans la feuille de calcul.

Plus loin avec la programmation…

STRUCTURES ALGORITHMIQUES

Permet d'activer une partie du code en fonction de la réalisation d'une condition ou pas.

Syntaxe

```
If condition Then
  bloc d'instructions
  si la condition est vraie
Else
  bloc d'instructions
  si la condition est fausse
End If
```

(1) Condition est souvent une opération de comparaison

(2) La valeur de retour de Condition est de type booléen (True ou False)

(3) Then doit être sur la même ligne que If

(4) La partie Else est facultative (ne rien faire si la condition est fausse)

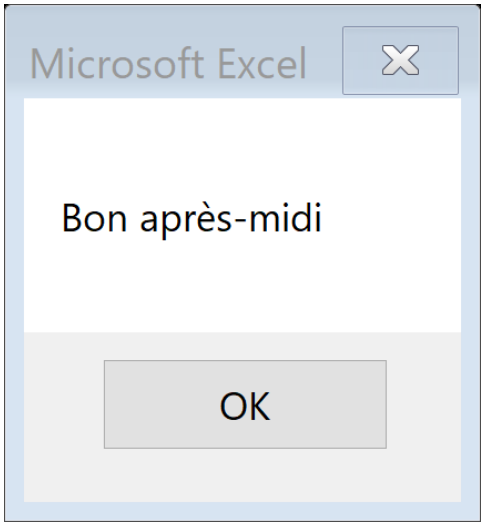
(5) Il est possible d'imbriquer une autre structure conditionnelle If dans les blocs d'instructions

Entrées : prix HT (réel), catégorie de produit (chaîne)
Sortie : prix TTC (réel)

```
Public Function MonTTCBis(pht As Double, cat As String) As Double
'déclarer la variable de calcul
Dim pttc As Double
'en fonction de la catégorie de produit
If (cat = "luxe") Then
    pttc = pht * 1.33
Else
    ‘la valeur de cat est différente de ‘’luxe’’
    pttc = pht * 1.2
End If
'renvoyer le résultat
MonTTCBis = pttc
End Function
```

Branchement conditionnel IF/ElseIf – Un exemple

```
Sub bonjour()

Dim msg As String

If Time < 0.5 Then

    msg = "jour"

ElseIf Time < 0.75 Then

    msg = "après-midi"

Else

    msg = "soir"

End If

MsgBox "Bon" & msg


End Sub
```



Permet d'activer une partie du code en fonction des valeurs prises par une variable de contrôle. Peut se substituer au IF, mais pas toujours, tout dépend de la forme de la condition (*condition composée, on doit passer par un IF*).

Syntaxe

```
Select Case variable
       Case valeur 1
              bloc d'instructions
       Case valeur 2
              bloc d'instructions
       ...
       Case Else
              bloc d'instructions
End Select
```

(1) Variable est la variable de contrôle, elle peut être de n'importe quel type en VBA, y compris un réel ou une chaîne de caractères

(2) Valeur doit être de type compatible avec variable

(3) La partie Case Else est facultative

(4) L'imbrication avec un autre IF ou un autre Select Case (autre variable de contrôle) est possible.

Entrées : prix HT (réel), catégorie de produit (chaîne)
Sortie : prix TTC (réel)

```
'fonction select case
Public Function MonTTCSelon(pht As Double, cat As String) As Double
'déclarer la variable de calcul
Dim pttc As Double
'en fonction de la catégorie de produit
Select Case cat
    Case "luxe"
        pttc = pht * 1.33
    Case Else
        pttc = pht * 1.2 'toute autre valeur que ‘’luxe’’
End Select
'renvoyer le résultat
MonTTCSelon = pttc
End Function
```

Il est possible d'introduire des plages de valeurs dans la partie Case de la structure Select Case. La comparaison devient plus sophistiquée. Variable est un numérique dans ce cas, entier ou même réel.

Syntaxe

```
Select Case variable
       Case Is op.de.comparaison valeur
              bloc d'instructions

       Case valeur de départ To valeur de fin
              bloc d'instructions

       Case Else
              bloc d'instructions
End Select
```

Entrée : quantité (entier)
Sortie : prix unitaire (réel)
Calcul : quantité < 100 → p.u. = 0.5
100 ≤ quantité ≤ 200 → p.u. = 0.3
quantité > 200 → p.u. = 0.2

```
'calcul du prix unitaire en fonction de la quantité
Public Function MonPU(quantite As Long) As Double
'variable intermédiaire
Dim pu As Double
'selon les valeurs de quantité
Select Case quantite
    Case Is < 100
        pu = 0.5
    Case 100 To 200
        pu = 0.3
    Case Is > 200 'Case Else aurait fait l'affaire aussi
        pu = 0.2
End Select
MonPU = pu
End Function
```

Faire répéter l'exécution d'un bloc d'instructions. Le nombre d'itérations est contrôlé par un indice.

Syntaxe

```
For indice = val.départ to val.fin step pas
        bloc d'instructions
        ...
Next indice
```

(1) Indice est un type ordonné, très souvent un numérique

(2) pas contrôle le passage d'une valeur à l'autre d'indice, si omis, pas = 1 par défaut

(3) Next entérine le passage à la valeur suivante de indice, si cette prochaine valeur est > à val.fin, on sort de la boucle

(4) Val.fin doit être superieure à val.départ pour que l'on rentre dans la boucle

(5) Si pas est négatif, val.fin doit être inférieure à val.départ cette fois-ci

(6) L'instruction Exit For permet de sortir prématurément de la boucle

(7) On peut imbriquer des boucles (une boucle à l'intérieur d'une autre boucle)

Entrée : n (entier)
Sortie : S (réel)
Calcul : $S = 1^2 + 2^2 + \ldots + n^2$

```
'calcul de la somme des carrés des valeurs
Public Function MaSommeCarre(n As Long) As Double
'variables de calcul (s pour la somme, i : indice)
Dim s As Double, i As Long
'initialisation
s = 0
'boucle avec l'indice i
For i = 1 To n Step 1
    s = s + i ^ 2
‘Next joue le rôle de l’incrémentation (i suivant)
Next i
'renvoyer le résultat
MaSommeCarre = s
End Function
```

Faire répéter l'exécution d'un bloc d'instructions. Le nombre d'itérations est contrôlé par une condition. Attention à la boucle infinie c.-à-d. la condition permettant de sortir de la boucle n'est jamais déclenchée.

Syntaxe

```
Do While condition
        Bloc d'instructions...
        ...
Loop
```

(1) Condition est un booléen, c'est souvent une opération de comparaison

(2) On continue l'exécution TANT QUE la condition est vraie ; si la condition est fausse, on sort de la boucle

(3) Exit Do permet de provoquer la sortie prématurée de la boucle

Si la condition est fausse d'emblée. On peut ne pas rentrer dans la boucle.

Boucle DO WHILE…LOOP (un exemple)

Entrée : n (entier)
Sortie : S (réel)
Calcul : $S = 1^2 + 2^2 + \ldots + n^2$

```
'calcul de la somme des carrés des valeurs
Public Function MaSommeCarreWhile(n As Long) As Double
'variables de calcul
Dim s As Double, i As Long
'initialisation
s = 0
'il nous revient aussi d'initialiser l'indice
i = 1
'boucle TANT QUE
Do While (i <= n)
    'sommer
    s = s + i ^ 2
    'pas de next, nous devons incrémenter l'indice
    i = i + 1
Loop
'renvoyer le résultat
MaSommeCarreWhile = s
End Function
```

Faire répéter l'exécution d'un bloc d'instructions. Le nombre d'itérations est contrôlé par une condition.

Syntaxe

```
Do
        Bloc d'instructions
        ...
        ...
Loop While condition
```

- On est sûr de rentrer au moins une fois dans la boucle. !
- Le choix de la bonne structure (Faire.. Tant Que ou Tant Que.. Faire) dépend du problème à traiter !

Les boucles DO contrôlées par une condition sont très riches en VBA.

```
Do { While | Until } condition
        [ statements ]
        [ Exit Do ]
        [ statements ]
Loop
-or-
Do
        [ statements ]
        [ Exit Do ]
        [ statements ]
Loop { While | Until } condition
```

Le Répeter… Jusqu'à (Until) existe aussi.

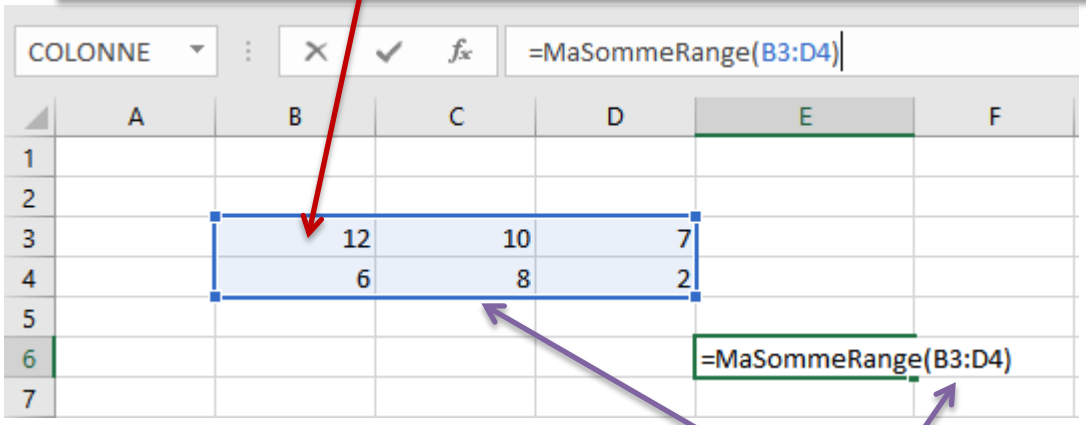
Le type « plage de cellules » spécifique à Excel

LE TYPE RANGE

Le type RANGE désigne une plage de cellules, c'est un type spécifique à Excel.

Exemple

Coin en haut et à gauche de la plage de cellules passée en paramètre de la fonction = coordonnée (1, 1) c.-à-d. ligne n°1 et colonne n°1, quelle que soit la position absolue de la plage dans la feuille de calcul (ici le coin nord-ouest est en B3)



Un bloc de cellules (B3:D4) est passé en paramètre de la fonction. Ce bloc est forcément rectangulaire, avec, ici : 2 lignes et 3 colonnes.

La fonction MaSommeRange() est censée faire la même chose que la fonction standard SOMME() d'Excel.

Entrée : plage (range)
Sortie : S (réel)
Calcul : Somme des valeurs

```
'Travail sur le type Range
Public Function MaSommeRange(plage As Range) As Double
'variables intermédiaires
Dim s As Double, i As Long, j As Long
'initialisation de la somme
s = 0
'parcours de la plage de cellules
For i = 1 To plage.Rows.Count Step 1 'lignes
    For j = 1 To plage.Columns.Count Step 1 'colonnes
        'lecture des valeurs et somme
        s = s + plage.Cells(i, j).Value
    Next j
Next i
'renvoyer le résultat
MaSommeRange = s
End Function
```

Lignes et colonnes commencent à l'indice 1, quelle que soit la position de la plage dans la feuille.

Nombre de lignes de la plage de cellules.

Nombre de colonnes.

Accès à la valeur (Value) de la cellule : ligne n°i, colonne n°j

La boucle For Each est adaptée au parcours des collections.

Or une plage de cellules est une collection de cellules.

```
'Travail sur le type Range avec un For Each
Public Function MaSommeRangeEach(plage As Range) As Double
'variables intermédiaires
Dim s As Double, cellule As Range
'initialisation de la somme
s = 0
'parcours de la plage de cellules
For Each cellule In plage
    s = s + cellule.Value
Next cellule
'renvoyer le résultat
MaSommeRangeEach = s
End Function
```

Une cellule est une plage de cellules avec une seule cellule.

plage fait figure de collection à traiter. Qu'importe le sens du parcours ici (ligne par ligne, ou colonne par colonne).

C'est bien la valeur contenue dans la cellule qui est exploitée pour la somme.

Type spécial qui peut contenir toutes sortes de valeur

LE TYPE VARIANT

Le type de variant peut gérer tout type de valeurs. Il est très souple, particulièrement commode quand on ne connaît pas à l'avance le type à utiliser. Mais attention, il ne faut pas en abuser, il est très lent parce que multiplie les vérifications à chaque accès à la variable correspondante.

Entrée : a, b (réel)
Sortie : a/b si b ≠ 0, « division par zéro » sinon

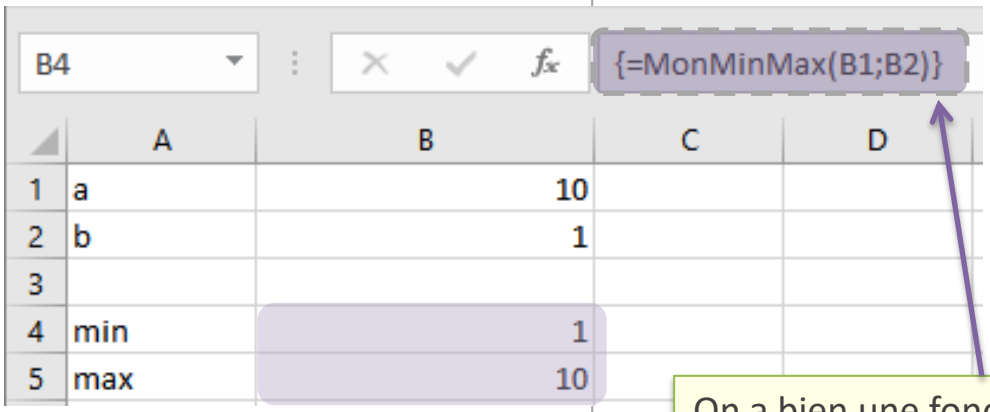
Un coup, la fonction renvoie un réel, un autre coup elle doit renvoyer une chaîne de caractères.

```
'utilisation du type variant
Public Function MaDivision(a As Double, b As Double) As Variant
'var. intermédiaire
Dim resultat As Variant
'calcul
If (b <> 0) Then
    resultat = a / b
Else
    resultat = "division par zéro"
End If
'renvoyer le résultat
MaDivision = resultat
End Function
```

Dans la même variable resultat, de type variant, on peut affecter un réel et une chaîne de caractères.

On peut s'en servir pour renvoyer un tableau. Une fonction peut donc renvoyer plusieurs valeurs d'un coup, à l'instar des fonctions matricielles d'Excel (il faut valider la saisie de la fonction avec la séquence de touches CTRL + MAJ + ENTREE).

```
'renvoyer plusieurs valeurs
Public Function MonMinMax(a As Double, b As Double) As Variant
'un tableau interne - matrice 2 lignes et 1 colonne
Dim tableau(1 To 2, 1 To 1) As Double
'identifier le min et le max
If (a < b) Then
    tableau(1, 1) = a
    tableau(2, 1) = b
Else
    tableau(1, 1) = b
    tableau(2, 1) = a
End If
'renvoyer le tableau
MonMinMax = tableau
End Function
```



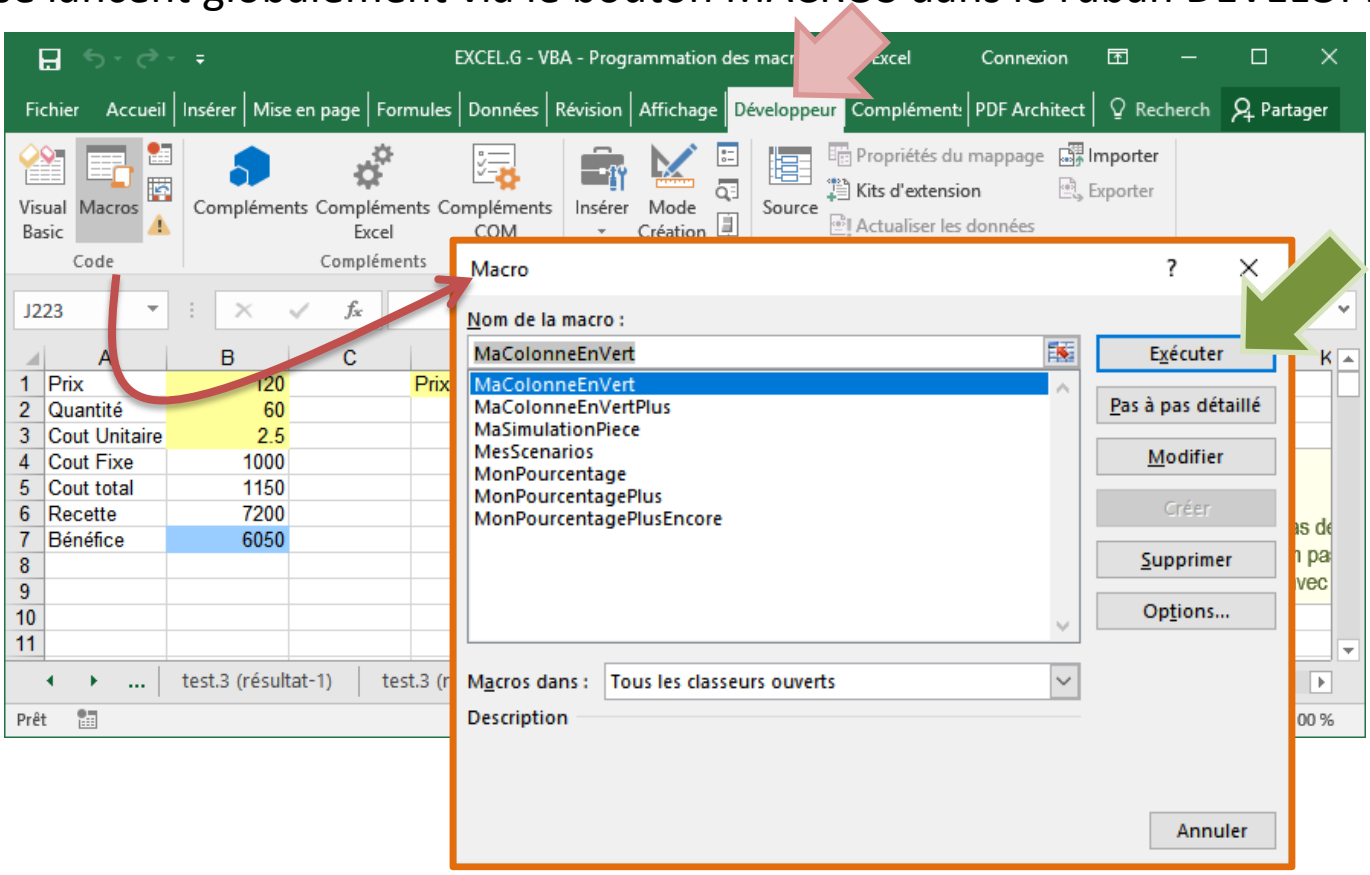
On a bien une fonction matricielle comme peuvent en témoigner les accolades { } qui encadrent l'appel de la fonction.

Programmation des macros – Travailler directement sur les feuilles

LES MACROS (1)

Les macros sont également des procédures que l'on crée à l'intérieur d'un module. Mais, à la différence des Function, ce sont des Sub() sans paramètres qui peuvent manipuler (accéder et modifier) directement les objets Excel (classeurs, feuilles, cellules, graphiques, scénarios, tableaux croisés dynamiques…).

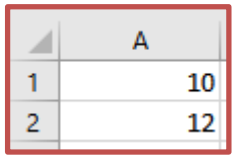
Ils ne s'exécutent pas de la même manière. Au lieu de les insérer dans une cellule, ils se lancent globalement via le bouton MACROS dans le ruban DEVELOPPEUR.



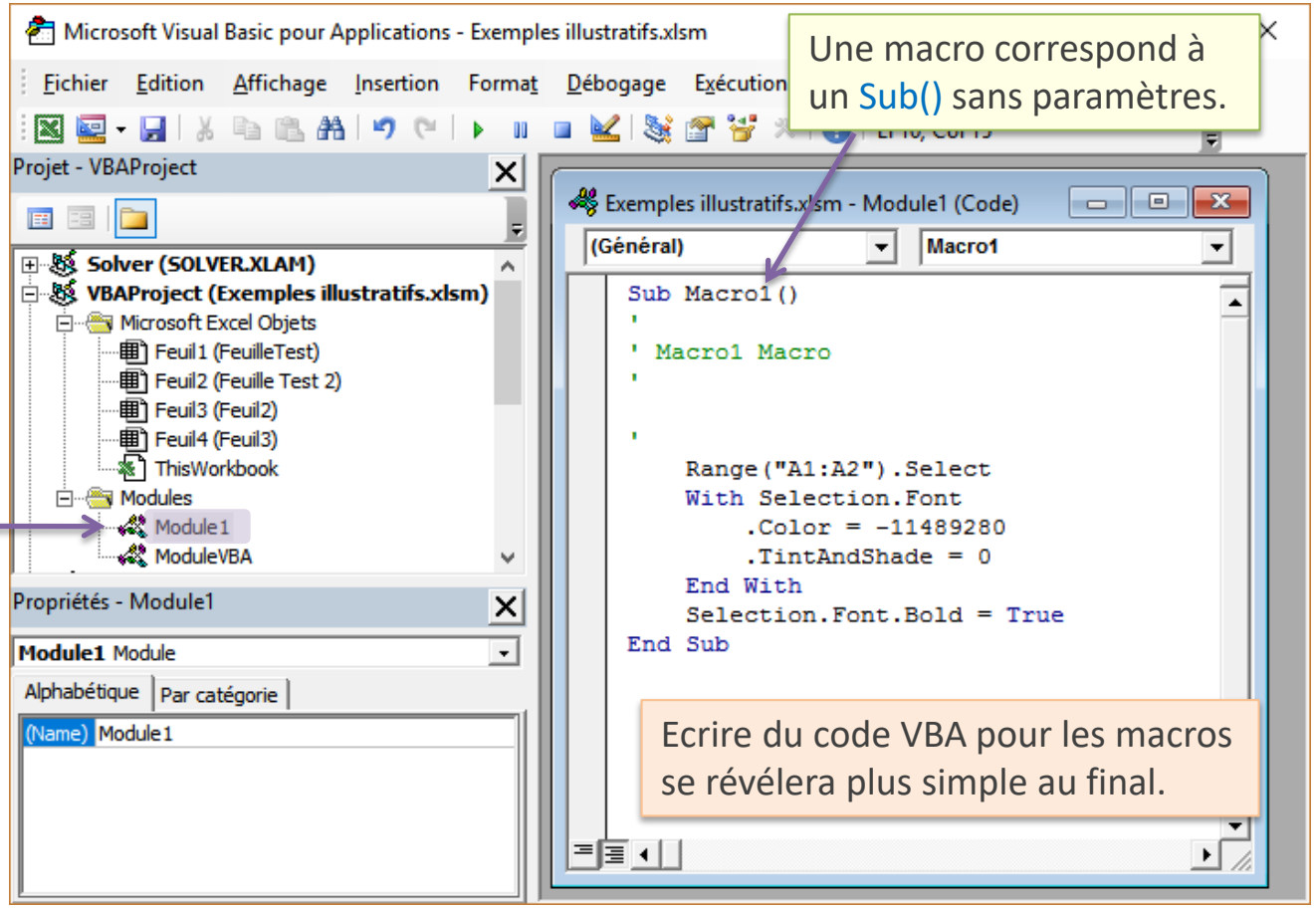
Une manière simple de générer une macro est de lancer l'enregistreur de macros.

Du code VBA est automatiquement généré.

Exemple : mettre en gras et vert le contenu des cellules A1 et A2



	A
1	10
2	12



Avantages :

- Il n'y a pas plus simple pour produire du code, on peut créer et exécuter une macro sans aucune notion de programmation
- Il nous donne des indications précieuses sur les commandes associées aux objets Excel

Inconvénients :

- On travaille à structure fixée, si la configuration de la feuille change, il n'est pas possible de lancer la macro
- On ne bénéficie pas de la puissance des structures algorithmiques

En définitive :

- Il peut nous aider à rédiger notre code en nous donnant des pistes sur la syntaxe des commandes et les objets adéquats à manipuler (ex. imprimer automatiquement des feuilles, on lance l'enregistreur une fois, on intègre son code dans le notre à l'intérieur d'une boucle).

Ecrire directement des macros est simple une fois assimilé la philosophie de l'approche, et identifié les principaux objets et l'accès à leurs propriétés et méthodes (l'enregistreur peut nous y aider).

Classeurs

```
Workbooks(''classeur1.xlsm'').Activate
```

Activer (sélectionner) le classeur dont le nom de fichier est ''classeur1.xlsm''

Feuilles

```
Sheets(''Feuil1'').Activate
```

Dans le classeur courant, activer la feuille de calcul dont le nom est ''Feuil1'' (visible dans la languette au bas de la feuille)

```
Workbooks(''classeur1.xlsm'').Sheets(''Feuil1'').Activate
```

On peut combiner les écritures.

Cellules

```
Cells(1,1).Value = 15
```

Dans la feuille courante du classeur courant, insérer la valeur 15 dans la cellule ligne n°1, colonne n°1 c.-à-d. en A1, les coordonnées sont absolues ici.

```
Sheets(''Feuil1'').Cells(1,1).Value = 15
```

De nouveau, on peut combiner.

Exemple de macros – Simulation valeurs de TVA

Ecrire une macro qui insère différentes valeurs de TVA en **B2** et récupère les valeurs de prix TTC en **B3**.

Les différentes valeurs de TVA testées doivent être retranscrites au fur et à mesure dans la colonne D.

Les valeurs de Prix TTC correspondantes doivent être recensées en colonne E

	A	B	C	D	E
1	PHT	100		TVA (%)	Prix TTC
2	TVA (%)	30			
3	PTTC	=B1*(1+B2/100)			
4					
5					
6					

```
Sub SimulationTVA()
'variables
Dim pht As Double, pttc As Double
Dim tva As Double
Dim i As Long
'début d'écriture des valeurs en ligne 2
i = 2
'récupérer la valeur du PHT
pht = Cells(1, 2).Value 'en B1
'faire varier la tva de 10% à 30% avec un pas de 5%
For tva = 10 To 30 Step 5
    'insérer la valeur de la TVA en B2
    Cells(2, 2).Value = tva
    'Récupérer le prix ttc en B3
    pttc = Cells(3, 2).Value
    'inscription des valeurs
    'TVA en colonne D
    Cells(i, 4).Value = tva
    'PTTC en colonne E
    Cells(i, 5).Value = pttc
    'passage à la ligne suivante
    i = i + 1
Next tva
End Sub
```

Remarque : il faut être sur la feuille adéquate avant de lancer la macro, sinon le programme ne saura pas où chercher **Cells(…).**

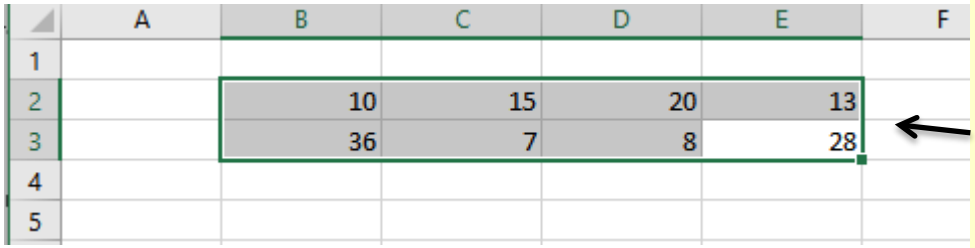
A l'issue de la simulation…

	A	B	C	D	E
1	PHT	100		TVA (%)	Prix TTC
2	TVA (%)	30		10	110
3	PTTC	130		15	115
4				20	120
5				25	125
6				30	130

Travailler sur les sélections de l'utilisateur

LES MACROS (2)

Comment programmer une macro qui manipule directement une plage de cellules sélectionnée par l'utilisateur ? Attention, nous ne sommes pas dans la même configuration que les fonctions personnalisées ici, nous n'insérons pas un résultat dans une cellule, nous manipulons et modifions directement la plage sélectionnée.



Exemple : dans cette sélection (les cellules doivent être sélectionnées avant de lancer la macro !), mettre en police verte les cellules contenant une valeur paire.

```
Sub MesValeursPaires()
'variable intermédiaire
Dim cellule As Range
'boucler sur la sélection
For Each cellule In Selection
    'tester le contenu
    If (cellule.Value Mod 2 = 0) Then
        'modifier la couleur de la police
        cellule.Font.ColorIndex = 4
    End If
Next cellule
End Sub
```

Selection est un objet Excel (Selection est donc un mot clé). Il est de type Range (que nous connaissons bien).

On peut aussi écrire Cellule.Font.Color=vbGreen

Résultat…

10	15	20	13
36	7	8	28

Dans une sélection, de nouveau les coordonnées sont relatives c.-à-d. le coin en haut à gauche d'une sélection correspond à la cellule (ligne n°1, colonne n°1) quelle que soit la position de la sélection dans la feuille.

!

```
Sub MesValeursPairesBis()
'variables intermédiaires
Dim i As Long, j As Long
'boucler sur les lignes
For i = 1 To Selection.Rows.Count
    'boucler sur les colonnes
    For j = 1 To Selection.Columns.Count
        'tester le contenu
        If (Selection.Cells(i, j).Value Mod 2 = 0) Then
            'modifier la couleur de la police
            Selection.Cells(i, j).Font.ColorIndex = 4
        End If
    Next j
Next i
End Sub
```

Aucun doute, Selection est bien de type Range

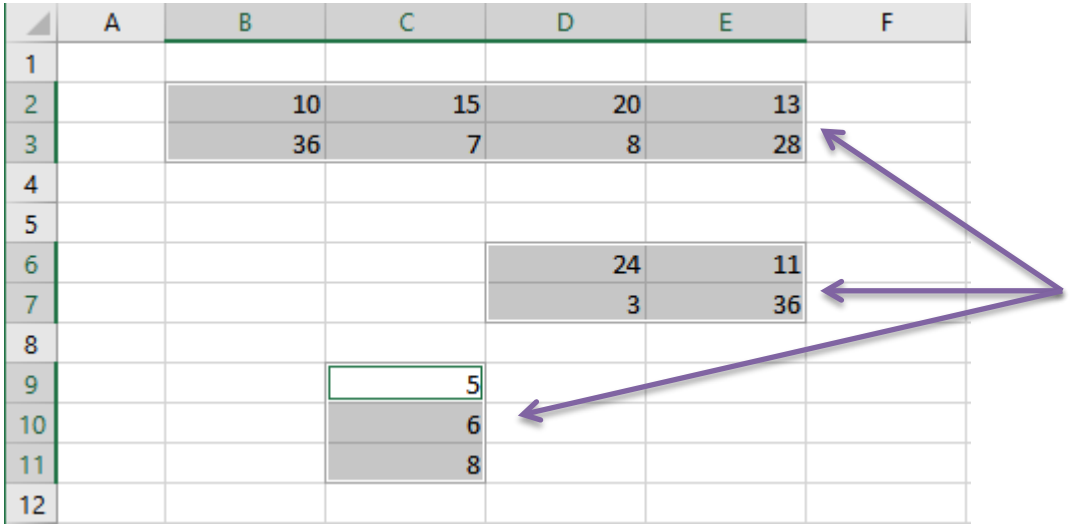
Identifier la première cellule contenant la valeur minimale dans une plage, mettre sa police en bleu.

10	15	20	13
36	7	8	28

```
Sub MonMinBleu()
'variables intermédiaires
'min va servir de cellule témoin
Dim cellule As Range, min As Range
'initialisation du témoin sur la 1ère cellule
Set min = Selection.Cells(1, 1)
'parcourir
For Each cellule In Selection
    'comparer avec le contenu de la cellule témoin
    If (cellule.Value < min.Value) Then
        'màj de la cellule témoin
        Set min = cellule
    End If
Next cellule
'mettre la couleur pour la cellule minimale
min.Font.ColorIndex = 5
End Sub
```

Range est un **objet**. Une affectation pour une variable objet doit être réalisée à l'aide de l'instruction Set

Une sélection peut être multiple aussi c.-à-d. contenant plusieurs ‘’zones’’



	A	B	C	D	E	F
1						
2		10	15	20	13	
3		36	7	8	28	
4						
5						
6				24	11	
7				3	36	
8						
9			5			
10			6			
11			8			
12						

Un exemple de sélection multiple avec 3 zones.

Très curieusement, le même mot clé Selection peut être exploité.

```
Selection.Areas.Count
```

Nombre de ‘’zones’’ dans la sélection.

```
Selection.Areas(k)
```

Accès à la zone n°k (qui est de type Range).
Areas est une collection de zones.

Sélection multiple – Un exemple

```
Sub MonMinZoneBleu()
'var. intermédiaires
Dim zone As Range, min As Range
'pour chaque zone
For Each zone In Selection.Areas
    'à l'intérieur de chaque zone
    'initialisation
    Set min = zone.Cells(1, 1)
    'parcours des cellules
    For Each cellule In zone
        'comparer
        If (cellule.Value < min.Value) Then
            'màj de la variable témoin
            Set min = cellule
        End If
    Next cellule
    'mettre la couleur pour la cellule minimale
    min.Font.ColorIndex = 5
'passage à la zone suivante
Next zone
End Sub
```

Pour chaque zone, mettre en police bleue la cellule contenant la valeur minimale.

Selection.Areas est une collection. On peut utiliser un For Each. On aurait pu aussi passer par un accès indicé. Par ex.

```
For k = 1 to Selection.Areas.Count
   Set zone = Selection.Areas(k)
   Etc…
```

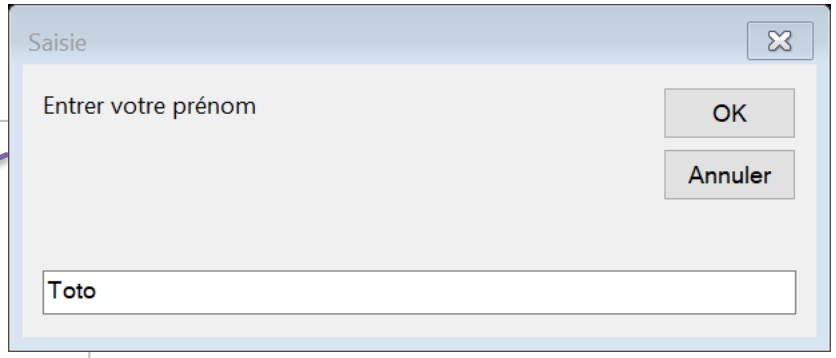
Résultat…

	A	B	C	D	E	F
1						
2		10	15	20	13	
3		36	7	8	28	
4						
5						
6				24	11	
7				3	36	
8						
9			5			
10			6			
11			8			
12						

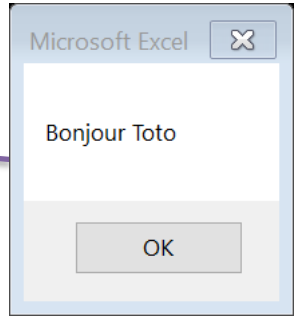
BOÎTES DE DIALOGUE

Les boîtes de dialogue permettent d'interagir avec l'utilisateur. Nous nous en tenons aux plus simples ici. InputBox() pour la saisie, MsgBox() pour l'affichage.

InputBox() est une fonction




```
Sub MesBoitesDeDialogue()
'var. intermédiaire
Dim prenom As String
'saisie
prenom = InputBox("Entrer votre prénom", "Saisie", "")
'affichage
MsgBox ("Bonjour " & prenom)
End Sub
```

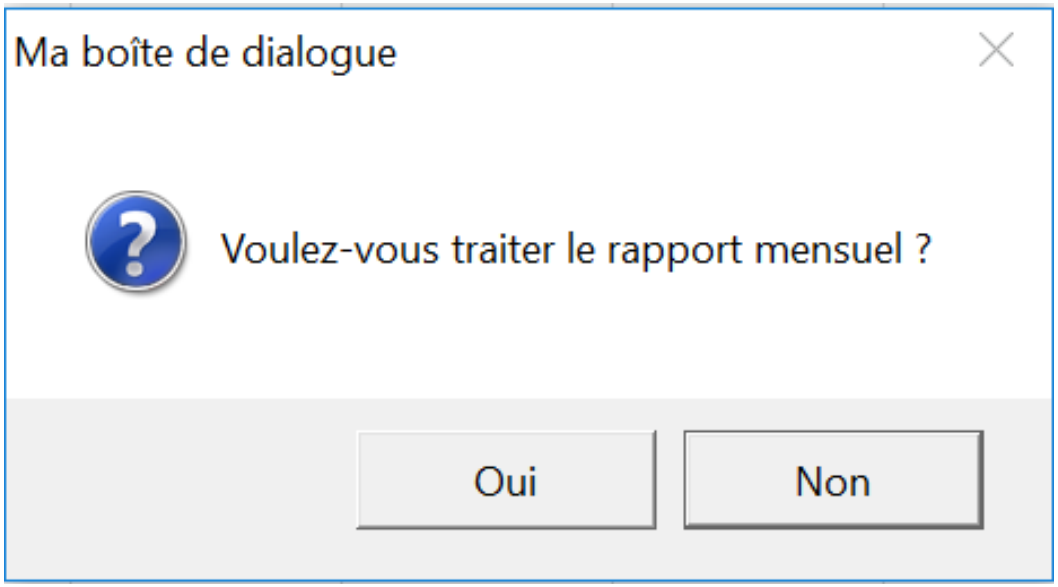


Noter la concaténation de chaînes de caractères pour faire apparaître le prénom dans la boîte de dialogue.

- Boite de dialogue de base, "personnalisable"

vbOKOnly	0	N'affiche que le bouton ok
vbOKCancel	1	Ok et Annuler
vbAbortRetryIgnore	2	Abandonner, Recommencer, Ignorer
vbYesNoCancel	3	Oui, Non, Annuler
vbYEsNo	4	Oui, Non
vbRetryCancel	5	Recommencer, Annuler
vbCritical	16	Icône message critique
vbQuestion	32	Icône Question
vbExclamation	48	Icône exclamation
vbInformation	64	Icône Information
vbDefaultButton1	0	Le premier bouton est par défaut
vbDefaultButton2	256	Le 2ième bouton est par défaut
vbDefaultButton3	512	Le 3ième bouton est par défaut
vbDefaultButton4	768	Le 4ième bouton est par défaut
vbSystemModal	4096	Suspend tout jusqu'à une réponse de l'utilisateur

```
Sub MaMsgBox()
Dim Config As Integer
Dim Reponse As Integer
Config = vbYesNo + vbQuestion + vbDefaultButton2
Reponse = MsgBox("Voulez-vous traiter le rapport mensuel ?", Config, "Ma boîte de dialogue")
If Reponse = vbYes Then
    MsgBox ("Youpi")
Else
    MsgBox ("Ohhhh")
End If
End Sub
```



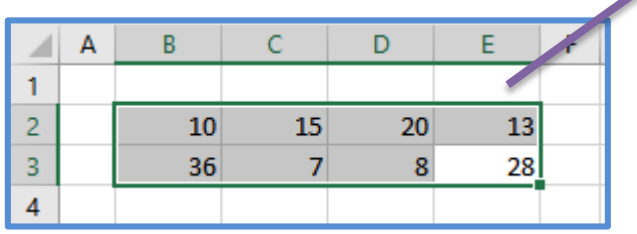
EXPLOITER LES FONCTIONS NATIVES D'EXCEL

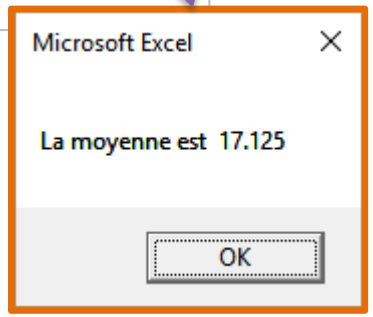
Excel dispose de fonctions natives puissantes. Nous pouvons y accéder dans nos programmes VBA.

Exemple : Vérifier qu'une sélection est simple (une seule zone), puis calculer et afficher la moyenne des valeurs dans une boîte de dialogue.

```
Sub MaMoyenneSelection()
'var. intermédiaire
Dim moyenne As Double
'vérifier la sélection
If (Selection.Areas.Count > 1) Then
    MsgBox ("Attention, ce n'est pas une sélection simple")
Else
    'faire calculer la moyenne de la sélection par Excel
    moyenne = Application.WorksheetFunction.Average(Selection)
    MsgBox ("La moyenne est " & Str(moyenne))
End If
End Sub
```

Noter la syntaxe. !





LES TABLEAUX

- Déclaration
  - Dim MonTableau(1 to 100) As Integer
  - Index débute à 0 par défaut;
    - Option Base 1

- Tableaux multidimensionnels
  - Dim MonTableau(1 to 10, 1 to 10) As Integer

- Affectation
  - MonTableau(3,4) = 125

- Tableaux dynamiques
- Création
  - Dim MonTableau() As Integer

- Redimensionnement
  - ReDim MonTableau(NombreElements)

- Redimensionner en gardant les données déjà présentes
  - ReDim Preserve MonTableau(NombreElements)

- **Structure pour afficher le contenu:**

```
Dim mois As Variant
Dim m As Variant
mois = Array("Janvier", "Mars", "Août", "Décembre")
For Each m In mois
        MsgBox m
Next m
```

- **Ou alors…**

```
Dim mois As Variant
Dim  i As Integer
mois = Array("Janvier", "Mars", "Août", "Décembre")
For i = 0 To 3
        MsgBox mois(i)
Next i
```

- Fonctions sur les tableaux:
  - Lbound : plus petit index du tableau
  - (Lbound,i) : plus petit index de la dimension i du tableau
  - Ubound : plus grand index
  - (Ubound,i) : plus grand index de la dimension i du tableau
  - Array(…) : retourne un tableau (doit être affecté à un Variant)
  - Erase : efface le tableau de la mémoire

- Exemples de fonctions de tableaux

```
Dim mois As Variant, i As Integer
mois = Array("Janvier", "Mars", "Août", "Décembre")

For i = LBound(mois) To UBound(mois)
        MsgBox mois(i)
Next i

Erase mois
```

Donne la borne inférieure

Donne la borne supérieure

Type structuré

Champs simples ou structurés

LES ENREGISTREMENTS

- Contrairement aux tableaux, ce type structuré permet de regrouper des données de types différents.

- Exemple : on identifie un ouvrage par un code, un titre, un ou plusieurs auteurs, un éditeur et éventuellement la date de parution.

- Ouvrage est une variable de type enregistrement; chacune de ces cinq données est un champ pouvant être simple ou structuré.

- Les enregistrements sont déclarés en VB avec le mot Type.

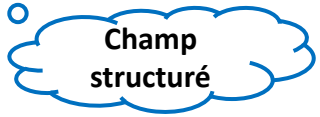
- Syntaxe :

```
Type NomEnregistrement
   Champ1 As type1
   Champ2 As type2
   …
End Type
```

- Exemple :


```
Type ouvrage
   code as Integer
   titre As String*40
   auteur As String*50
   editeur As String*50
   dateparution As Madate
End Type
```

```
Type Madate
   jour As Integer
   mois As Integer
   annee As Integer
End Type
```



- Exemple :

```
Type ouvrage
   code as Integer
   titre As String*40
   auteur As String*50
   editeur As String*50
   dateparution As Madate
End Type
```

```
Type MaDate
   jour As Integer
   mois As Integer
   annee As Integer
End Type
```

- Pour accéder à un champ :

```
Dim livre  As ouvrage
livre.auteur = "Durand "
livre.dateparution.annee = 1980
 'on s'aperçoit ici que l'on pourrait remplacer livre par un tableau
    dans le type
 ouvrage…Dim livre(1 To 10000) as ouvrage…
 livre(9).auteur = "Durand" s'il s'agit du neuvième livre de la liste…
…
```

Un étudiant est défini par son nom, son prénom, sa date de naissance et sa note :

```
Private Type Etudiant
        nom As String * 40
        prenom As String * 40
        dateNaissance As Date
        note As Double
End Type
```

Une classe peut contenir au plus 30 étudiants :

```
Const NbMax = 30                          'pour le nombre limite d'étudiants
Private Type Classe
    liste(NbMax) As Etudiant              'la liste est un tableau d'étudiants
    nbr As Integer                        'le nombre réel des étudiants
End Type
```

On déclare ensuite la classe d'étudiants :

```
Dim c As Classe
```

- L'exemple précédent sera complété dans le prochain cours sur les interfaces graphiques…
- Comment définir une matrice ?
- Créer un programme qui affiche le nombre de lignes et de colonnes d'une matrice saisie sur la Feuil1 du classeur.

LES CHAÎNES DE CARACTÈRES

- Concaténation: & ("fabrice" & " pasquier")
- Construction périodique: String(20, "x")
  - Répète 20 fois le caractère 'x'
  - Space(10): génère une séquence de 10 espaces
- Eclatement: Split(chaine, séparateur)
  - s = Split("c:\windows\system32\driver.dll", "\")
  - s doit être de type Variant

```
Sub eclatetout()

    Dim Tableau() As String
    Dim i As Integer

    'découpe la chaine en fonction des espaces " "
    'le résultat de la fonction Split est stocké dans un tableau
    Tableau = Split("c:\windows\system32\driver.dll", "\")

    'boucle sur le tableau pour visualiser le résultat
    For i = 0 To UBound(Tableau)
        'Le résultat s'affiche dans la fenêtre d'execution de l'éditeur de macros
        Debug.Print Tableau(i)
    Next i

End Sub
```

- Longueur: Len(chaîne)

- Positionnement: InStr(chaîne, caractère)
  - pos = InStr("Il fait beau", "b") retourne 9

- MAJ, min:
    - LCase("BonjouR") retourne "bonjour"
    - UCase("BonjouR") retourne "BONJOUR"

- Sélection de caractères: Mid, Left, Right
    - Left("Fabrice", 3) retourne "Fab"
    - Right("Fabrice", 3) retourne "ice"
    - Mid("Fabrice", 5, 2) retourne "ic"

- La fonction Format(…) retourne une chaîne de caractères formatée en fonction des paramètres
- Même fonctionnement que dans Excel
- Format(12121.13, "##'###.00") retourne 12'121.13
- Format("salut", "<") ⇔ UCase("salut")
- Format(Date, "yy/mmmm/dd")
  - La fonction Date retourne la date actuelle. Celle-ci doit être formatée avant affichage dans une boîte de dialogue, sinon elle sera affichée sous la forme spécifiée dans les options régionales (dd/mm/yy)
- D'autres fonctions ici : https://silkyroad.developpez.com/VBA/ManipulerChainesCaracteres/

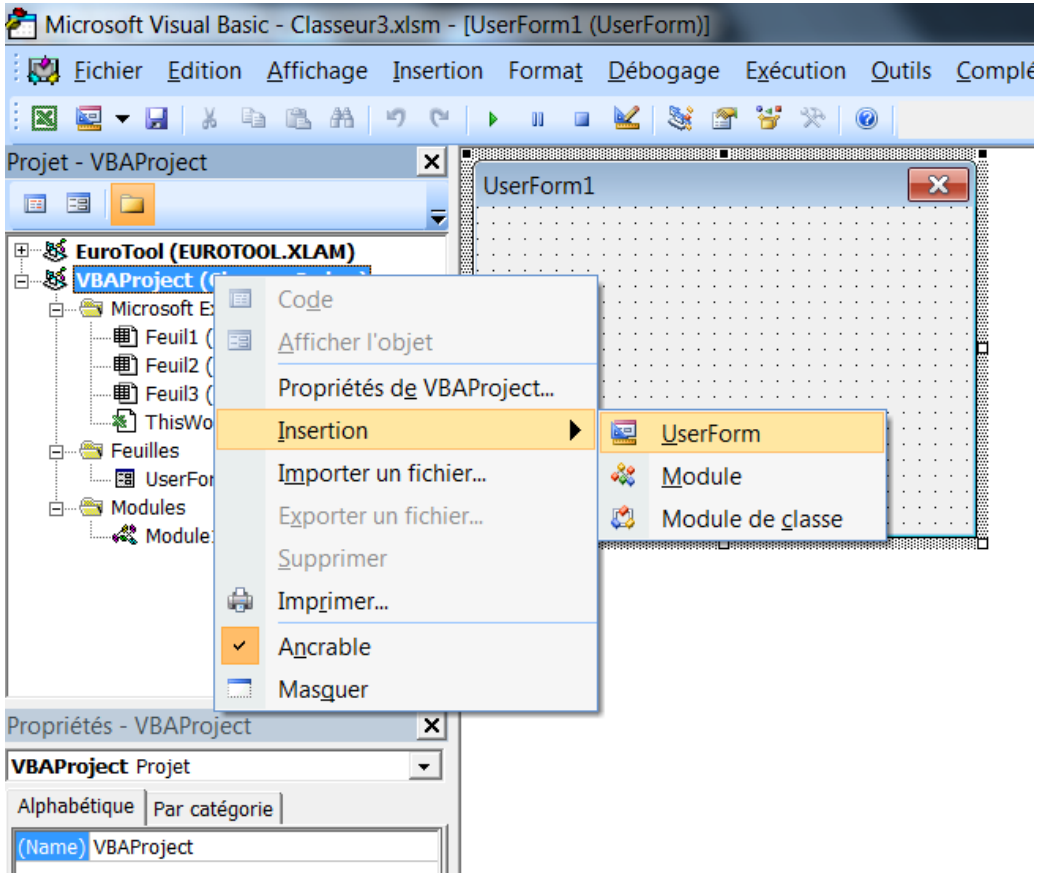
UserForm et éditeur graphique

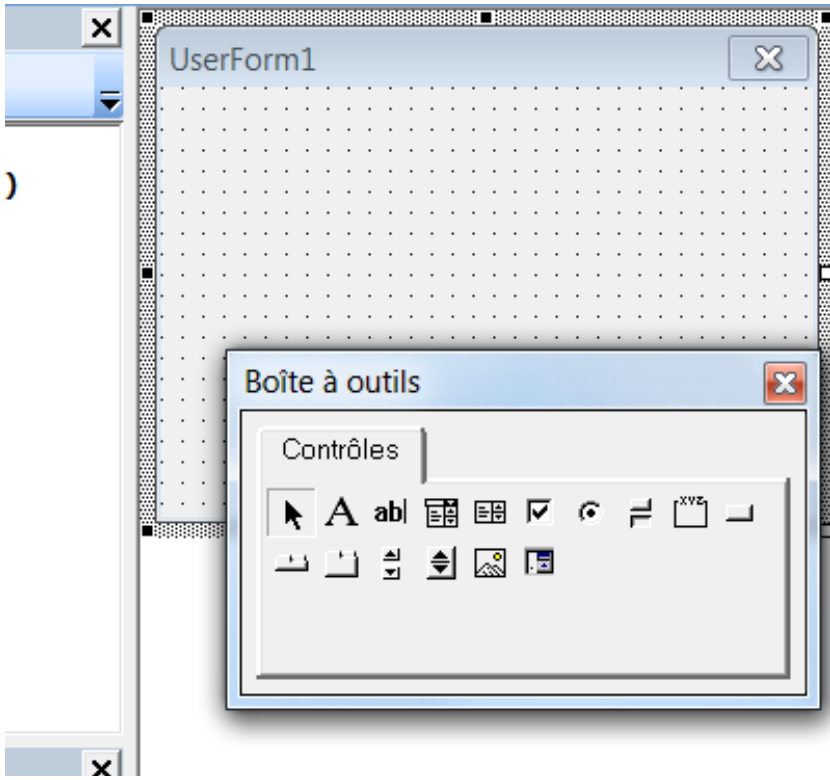
Les différents contrôles

DÉVELOPPEMENT RAPIDE D'INTERFACES

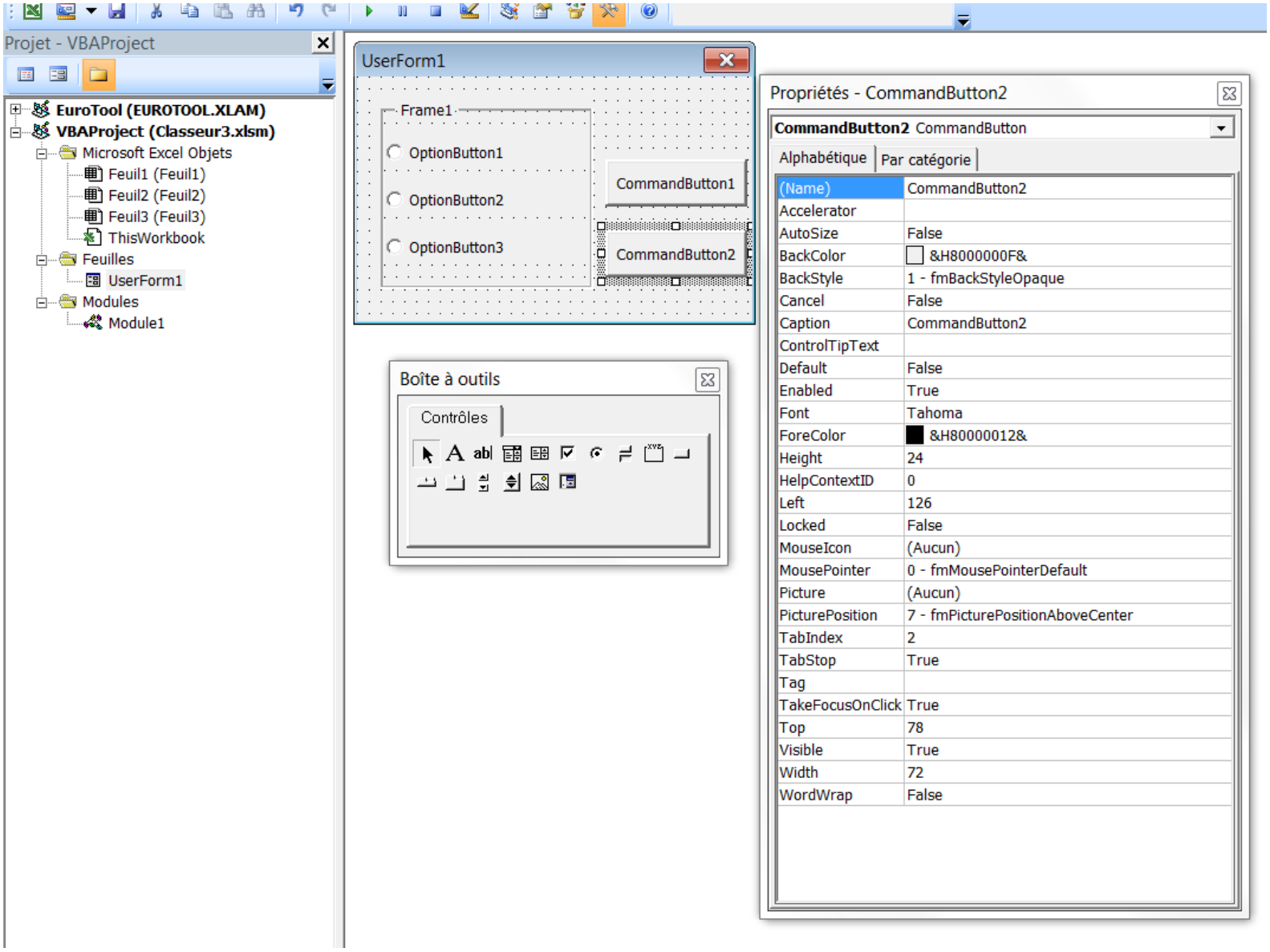
- 1) Imaginer la boîte de dialogue: à quoi sert-elle, où sera-t-elle utilisée?
- 2) Créer un nouvel objet userForm dans l'éditeur VBE
- 3)Ajouter des contrôles
  - Zones de textes
  - Boutons radio
  - Cases à cocher
  - Listes
- 4) Modifier les propriétés des éléments
- 5) Ecrire les procédures d'évènements des différents contrôles
- 6)Ecrire la procédure affichant la boîte de dialogue.

L'éditeur graphique de USerForm

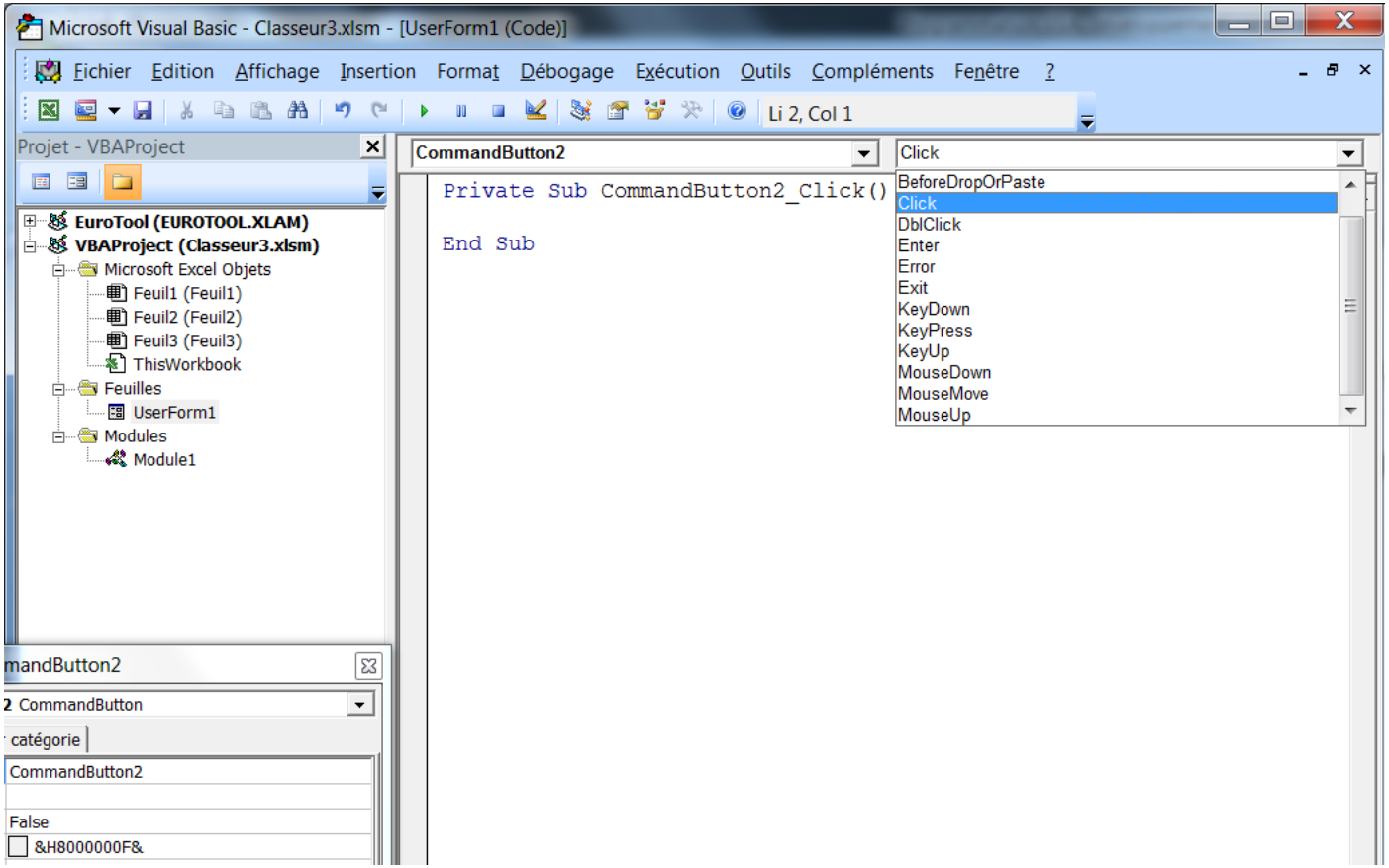




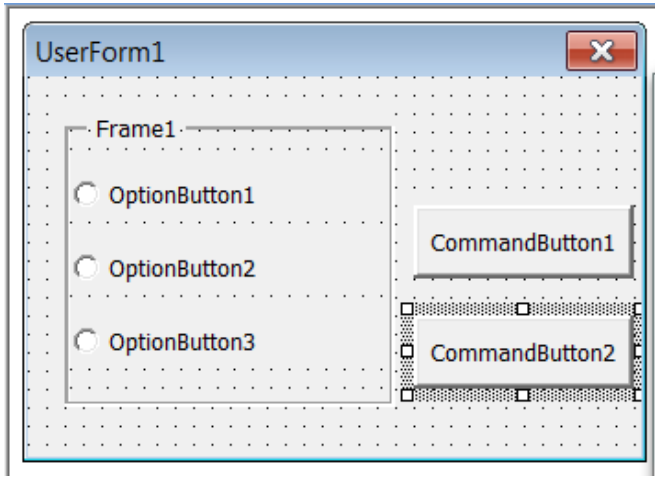
Editer les propriétés des contrôles



- Double-cliquer sur le contrôle dont on veut éditer les évènements



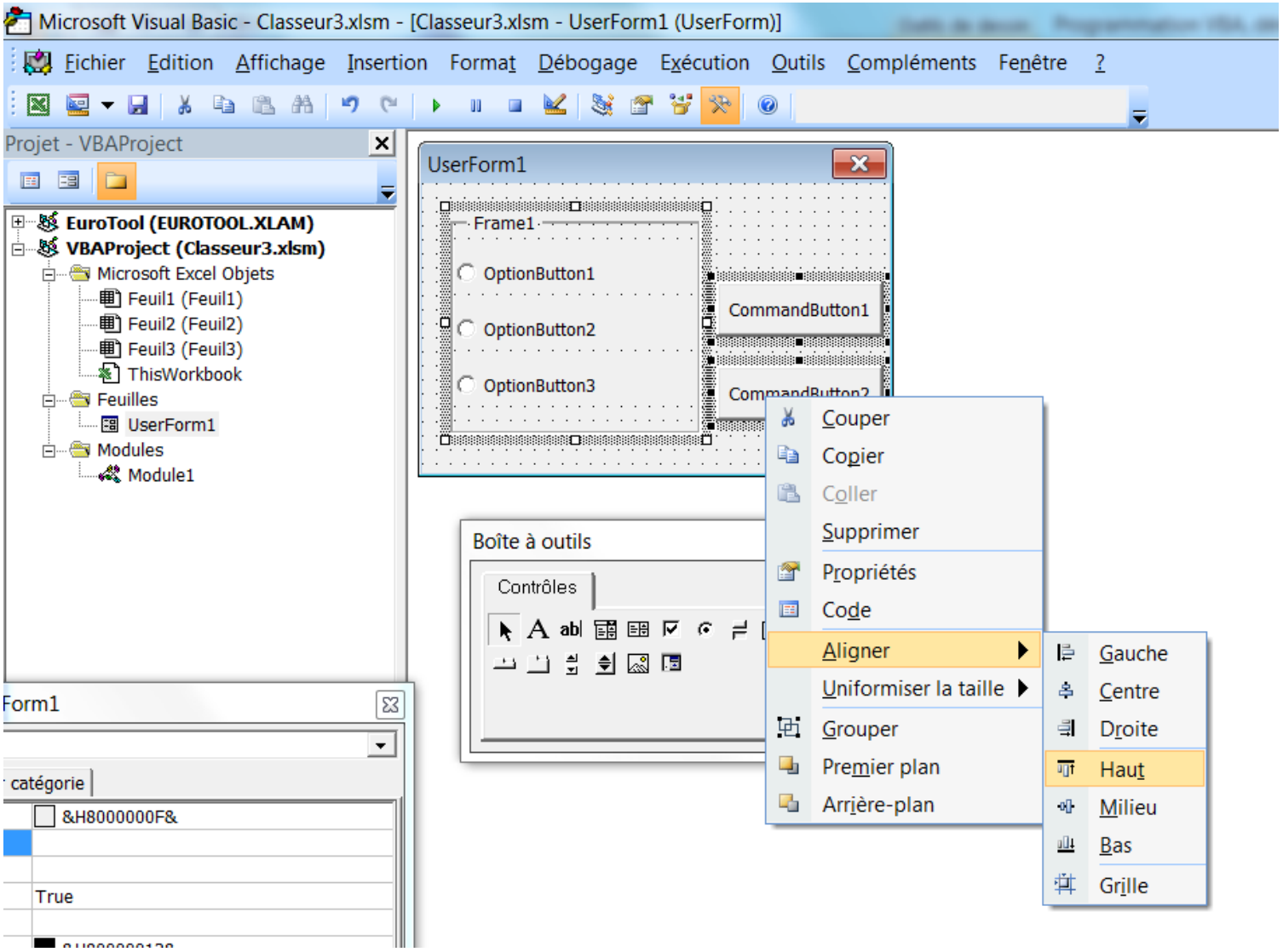
- La case à cocher
  - Accelerator
  - Value
- Zone de liste modifiable
  - ListRow
  - RowSource
  - Value
- Bouton
  - Annuler
  - Default
- Image
  - picture



- Multipage: faire des onglets.
- Bouton d'option (bouton radio): sélection d'UNE option parmi plusieurs.
  - Un groupe est défini par tous les boutons ayant la même propriété GroupName ou si tous les boutons sont dans un même cadre.
- RefEdit: permettre à l'utilisateur de sélectionner une plage dans une feuille de calcul
- Barre de défilement: ascenceur permettant de définir/ afficher une valeur

- Contrôle Toupie: 2 boutons fléchés permettant d'incrémenter / décrémenter une valeur
- Contrôle zone de texte: insérer du texte!
- Bouton bascule: similaire à la case à cocher

Dimensionner / Aligner les contrôles



HIÉRARCHIE ET APPLICATION ...COMPLÉMENT

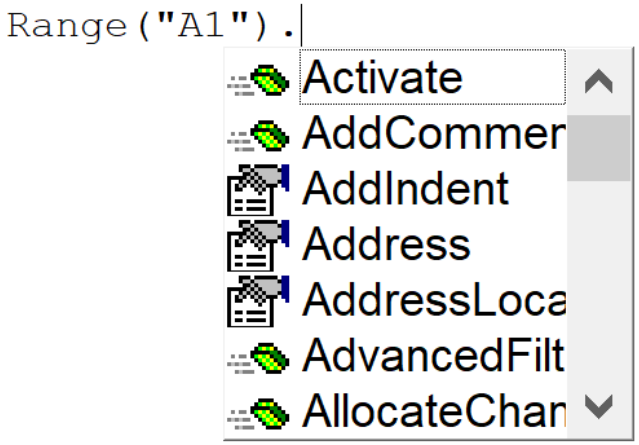
- On écrit une hiérarchie en VB avec des . (point)
- On se réfère à la cellule A1 de la 1ère feuille de calculs:
  - Excel.Workbooks(1).Sheets(1).Cells(1,1)
- Noter que l'index 1 est le plus petit. Pas de 0.
- Workbooks(1) => on prend le 1er élément de la collection des Workbook.
- Des simplifications d'écrire sont possibles
  - Sheets(1).Cells(1,1) est équivalent

- Cells est également une collection. Chaque élément de la collection est une cellule, indexée matriciellement.
- Cells(3, 4) est la cellule (Cell) "C4".
- Cependant, l'objet Cell n'existe pas vraiment en Excel, il est remplacé par Range.
- Chaque Range a de nombreuses propriétés, comme la couleur ou la police.

- Ecrire dans la cellule A1:
  - Excel.Workbooks(1).Sheets(1).Cells(1, 1).Value = "salut"
  - Excel.Workbooks(1).Sheets(1).Range("A1").Value = "salut"
  - Sheets(1).Range("A1").Value = "salut"
  - Range("A1").Value = "salut" (il faut être sur que la 1ère feuille de calculs est sélectionnée dans Excel…)

- Les 2 dernières solutions sont des simplifications d'écriture, mais évidemment moins précises…

- Lorsque l'on tape Range("A1"). Visual Basic propose toute une liste de méthodes et de propriétés disponibles pour cet objet.



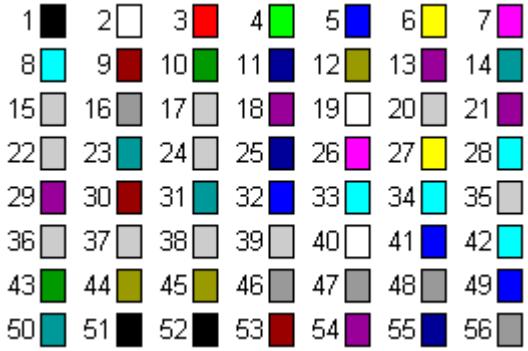
- Une méthode est une action que l'on peut exécuter sur un objet.

- On peut changer la police ainsi:
  - Gras:
    - Sheets(1).Range("A1").Font.Bold = True
  - Taille:
    - Sheets(1).Range("A1").Font.Size = 12
  - Nom:
    - Sheets(1).Range("A1").Font.Name = "Arial"

- Ecrire une valeur:
  - Sheets(1).Range("A1").Value = 12

- Ecrire une formule:
  - Sheets(1).Range("A1").Value = "=SUM(A1:B1)"
    - Attention: il est nécessaire d'écrire la formule en anglais!

- 2 syntaxes pour les couleurs:
  - Sheets(1).Range("A1").Interior.Color = vbRed
    - (vbBlack, vbRed, vbGreen, vbYellow, vbBlue, vbMagenta, vbCyan, vbWhite ou valeurs en hexa)
  - Sheets(1).Range("A1").Interior.ColorIndex



- Définition de la couleur du bord:
  - Sheets(1).Range("A1").Borders.Color = vbRed

Ou

  - Sheets(1).Range("A1").Borders.ColorIndex = 13

- Système de couleur RGB
- 3 valeurs codées de 0 à 255 (3 x 8bits = 24bits) (Rouge, Vert, Bleu)
- Blanc : R:255 G:255 B:255
  Bleu: R:0 G:0 B:255
- Convertir en hexadécimale
  - 255->FF 15->0F
  - Ecrire les 3 valeurs à côté: 0xFFED10
  - Ecrire en décimale 0xFFED10 ->1677238

PROGRAMMATION OBJET…LES MODULES DE CLASSE

- D'abords les enregistrements :
  - Regroupement au sein d'une même structure d'un ensemble de données élémentaires.
  - Déclarés en VB avec le mot clé Type.

- Syntaxe :

```
Type NomEnregistrement
   Champ1 As type1
   Champ2 As type2
   …
End Type
```

- Exemple :

```
Type Etudiant
   numero as Long
   nom As String
   prénom As String
   dateNaissance As Madate
End Type
```

```
Type Madate
   jour As Integer
   mois As Integer
   annee As Integer
End Type
```

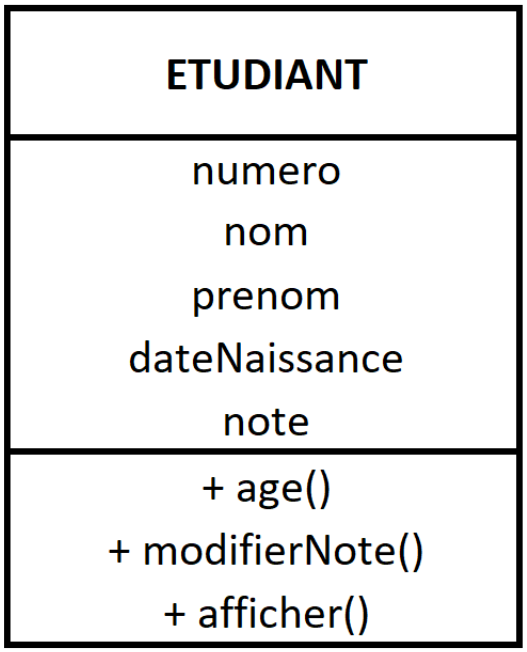
- Pour accéder aux champs

```
Dim e As Etudiant
…
e.numero = 100
e.dateNaissance.annee = 1988
```

- Enrichissement de la notion d'enregistrement.
- Objet = ensemble de données (attributs) permettant de
  - caractériser l'objet

+

  - ensemble de programmes (méthodes) servant à modifier les attributs.
- Programme Objet = ensemble d'objets.


ETUDIANT
numero
nom
prenom
dateNaissance
note
+ age()
+ modifierNote()
+ afficher()

- Consiste à regrouper dans un objet ses propriétés et les méthodes qu'on peut lui appliquer.

- Intérêts :

  - Masquer l'implémentation : les utilisateurs n'ont pas besoin de savoir comment vous avez implémenté votre module.

  - Ces objets, une fois implémentés, deviennent des « boîtes noires ».

- La méthode la plus simple pour implémenter des propriétés est d'utiliser des variables publiques. Pour implémenter une propriété, il suffit de déclarer des variables publiques dans notre module de classe

```
Public nom as String
Public prénom as String
…
```

- Problèmes :
  - Impossible de créer une propriété en lecture (ou écriture) seule.
  - Impossible de savoir quand une propriété est modifiée.
  - Impossible de vérifier la validé des valeurs (par exemple une date de naissance doit toujours être une date révolue).

- Les procédures Property permettent de donner un accès complet ou limité aux propriétés d'un objet.

- Nous accédons à ces données à travers des méthodes.

- Il existe 3 procédures Property :

  - Property get : lecture de la propriété quel que soit son type.
  - Property let : écriture des propriétés de type simple.
  - Property set : écriture des propriétés de type objet.

Les Classes : les procédures Property…exemple

```
Private eNumero As Long
Private eNom As String
Private ePrenom As String
Private eNote As Double
Private eDateNaissance As Date

Property Let numero(nNumero As Long)

If nNumero <= 0 Then

    MsgBox "Un numéro ne peut être négatif ou nul"

Else

    eNumero = nNumero

End If


End Property

Property Get numero() As Long

    numero = eNumero

End Property

Property Let nom(nNom As String)

If Len(nNom) = 0 Then

    MsgBox "Le nom ne peut être vide"

Else

    eNom = nNom

End If

End Property

Property Get nom() As String

    nom = eNom

End Property
```

- Comme dans un module simple, les méthodes peuvent, soit ne pas renvoyer de valeur (procédure Sub), soit renvoyer une valeur (fonction Function).

- Exemples :

```
Public Function age() As Integer
    age = DateDiff("yyyy", eDateNaissance, Now)
If DateAdd("yyyy", age, eDateNaissance) > Now Then
    age = age - 1
End If
End Function
```

```
Public Sub nSet(newNote As Double)
    eNote = newNote
End Sub
```

- Aller sur l'éditeur VBA (Alt+F11).
- Créer une nouvelle classe Etudiant.
- Saisir le code de la classe :
  1. D'abord les propriétés :
     - Private eNumero As Long
     - Private eNom As String
     - Private ePrenom As String
     - Private eDateNaissance As Date

1. D'abord les propriétés : …
2. Puis les accesseurs :

```
Property Let prenom(nPrenom As String)
If Len(nPrenom) = 0 Then
    MsgBox "Le prénom ne peut être vide"
Else
    ePrenom = nPrenom
End If
End Property
```

```
Property Get prenom() As String
    prenom = ePrenom
End Property
```

```
Property Let dateNaissance(nDateNaissance As Date)
If nDateNaissance > Now Then
    MsgBox "Une date de naissance ne peut être dans le futur !"
Else
    eDateNaissance = nDateNaissance
End If
End Property
```

```
Property Get dateNaissance() As Date
    dateNaissance = eDateNaissance
End Property
```

1. D'abord les propriétés : …
2. Puis les accesseurs : …
3. Et enfin les méthodes :

```
Public Function age() As Integer
    age = DateDiff("yyyy", eDateNaissance, Now)
If DateAdd("yyyy", age, eDateNaissance) > Now Then
    age = age - 1
End If
End Function
```

```
Public Sub nSet(newNote As Double)
    eNote = newNote
End Sub
```

- Pour tester notre classe :

  1. Dans l ’éditeur, on crée un nouveau module.
  2. On saisit le code du test :

```
Sub TestEtudiant()

Dim e As New Etudiant

With e
    .numero = InputBox("Numéro de l'étudiant :")
    .nom = InputBox("Nom de l'étudiant :")
    .prenom = InputBox("Prénom de l'étudiant :")
    .dateNaissance = InputBox("Date de naissance de l'étudiant :")
    .note = InputBox("Note de l'étudiant :")
End With

MsgBox ("L'étudiant(e) " + e.nom + " est âgé(e) de : " + CStr(e.age) + _
" ans" + ", il/elle a obtenu la note " + CStr(e.note))

End Sub
```

LES STRUCTURES COMPLEXES
LES PILES ET LES FILES

- Analogie de la pile d'assiettes
  - Last In First Out (LIFO)
- Opérations possibles
  - Insérer un élément dans une pile
  - Supprimer un élément d'une pile
  - Élément du sommet de la pile
  - Création d'une pile vide
  - Tester si une pile est vide

- Plusieurs façons de faire :
  - En particulier, à l'aide d'un tableau :
    - Le nombre max d'éléments dans la pile
    - Le contenu de la pile
    - Un indice pour pointer sur le sommet de la pile

- Type de données :

```
Const NMAX=30

Type TPile
        contenu(NMAX) as Integer
        sommet As Integer
End Type
```

```
Function PileVide(p As TPile) As Boolean
        If (p.sommet = -1) Then
                PileVide = True
        Else
                PileVide = False
        End If
End Function
```

```
Function PilePleine(p As TPile) As Boolean
        If (p.sommet = NMAX - 1) Then
                PilePleine = True
        Else
                PilePleine = False
        End If
End Function
```

```
Sub Empiler(p As TPile,elt As Integer)
    If (PilePleine(p) = False) Then
        p.sommet = p.sommet +1
        p.contenu(p.sommet) = elt
    Else
        MsgBox(''La pile est pleine !'')
    End If
End Sub
```

```
Sub Depiler(p As TPile)
    If (PileVide(p) = False) Then
        p.sommet = p.sommet - 1
    Else
        MsgBox(''La pile est vide !'')
    End If
End Sub
```

```
Function sommet(p As TPile) As Integer
    If (PileVide(p) = False) Then
        sommet = p.contenu(p.sommet)
    Else
        MsgBox(''La pile est vide !'')
    End If
End Function
```

- Analogie de la file d'attente
  - First In First out (FIFO)
- Opérations principales
  - Insertion d'un élément
  - Suppression d'un élément (le plus ancien de la file)
  - Quel est l'élément le plus ancien de la file ?
  - Création d'une file vide
  - Est-ce qu'une file est vide ?

- Plusieurs façons de faire :
  - En particulier, à l'aide d'un tableau :
    - Le nombre max d'éléments dans la file
    - Le contenu de la file
    - Un indice début qui pointe sur l'élément le plus ancien de la file
    - Un indice fin qui pointe sur le dernier élément inséré dans la file

- Type de données :

```
Const NMAX=30

Type TFile
        contenu(NMAX) as Integer
        debut As Integer
        fin As Integer
End Type
```

```
Function FileVide(f As TFile) As Boolean
        If (f.debut = f.fin) Then
                FileVide = True
        Else
                FileVide = False
        End If
End Function
```

```
Function FilePleine(f As TFile) As Boolean
        If (f.debut=(f.fin + 1) mod NMAX) Then
                FilePleine = True
        Else
                FilePleine = False
        End If
End Function
```

```
Sub Enfiler(f As TFile,elt As Integer)
    If (FilePleine(p) = False) Then
        f.contenu(f.fin) = elt
        f.fin = (f.fin +1) mod NMAX
    Else
        MsgBox(''La file est pleine !'')
    End If
End Sub
```

```
Sub Defiler(f As TFile)
    If (FileVide(f) = False) Then
        f.debut = (f.debut+1) mod NMAX
    Else
        MsgBox(''La file est vide !'')
    End If
End Sub
```

```
Function Tete(f As TFile) As Integer
    If (FileVide(f) = False) Then
        Tete = f.contenu(f.debut)
    Else
        MsgBox(''La file est vide !'')
    End If
End Function
```

- Créer un module pour simuler les piles.
- Votre module doit contenir la procédure suivante :

```
Sub main()
        Dim p As TPile
        Dim i As Integer
        i = 1
        While PilePleine(p)=False
                Call Empiler(p,i)
                i = i + 1
        WEnd
        While PileVide(p)=False
                MsgBox(Sommet(p))
                Call Depiler(p)
        Wend
End Sub
```

- Une personne est définie par un numéro, un nom et un prénom.
- Définir une structure PFile correspondant à une file de personne.
- Adapter les différentes procédures et fonctions pour qu'elles manipulent des files de personnes.

- A pratiquer en TP et sur le projet !

De la documentation à profusion (**inutile d'acheter des livres sur VBA**)

Site de cours de Microsoft

VBA sous Excel : https://msdn.microsoft.com/fr-fr/library/office/ee814737(v=office.14).aspx

Structures de décision : https://msdn.microsoft.com/fr-fr/library/hh892482(v=vs.90).aspx

Structures de boucle : https://msdn.microsoft.com/fr-fr/library/ezk76t25(v=vs.90).aspx

Autres cours et supports

Le Compagnon Info : http://www.lecompagnon.info/excel/

Excel Easy : http://www.excel-easy.com/

Cours VBA Gratuit : http://www.excel-pratique.com/fr/vba.php

Excel VBA for Complete Beginners : http://www.homeandlearn.org/index.html

Et d'autres très nombreux encore… faites chauffer les moteurs de recherche.

Merci

Hervé Hocquard (hocquard@labri.fr)

http://www.labri.fr/perso/hocquard/Teaching.html

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/