Programmation VBA

Attention !

- **Tout ce qui est dans ce cours est *simplifié* !**
  - Et par conséquent *inexact* !
- **On ne cherche pas l'efficacité !**
  - Certains exemples ici sont inefficaces
  - Ils servent surtout à illustrer certains concepts

Plan du cours

- **Visual Basic for Applications**
- **Dans la macro il y a...**
- **Diviser pour régner**
- **Le problème derrière le clavier**

Visual Basic for Applications

- **Visual Basic for Applications**
  - « VBA »
  - Caractéristiques
    - Interprété, impératif
    - Orienté objet
    - Événementiel
    - Typage hybride
  - Autres
    - Interfacé directement avec les applications bureautiques de la suite Office

Visual Basic for Applications

- **Utilité**
  - Automatiser les tâches,
  - Créer des applications complètes,
  - Sécuriser saisies et documents,
  - Créer de nouveaux menus et de nouvelles fonctions pour compléter le logiciel.
- **Comment ?**
  - Macro-commandes pré-programmées,
  - Écriture de macro-commandes personnalisées,
  - Écriture de fonctions personnalisées

Visual Basic for Applications

- **Dans quels outils peut-on utiliser VBA**
  - Excel
  - Word
  - PowerPoint
  - Access
  - Et certainement d'autres …
- **Chaque outil dispose de ses propres API**
  - API : Applicative Programmation Interfaces
  - Ensemble de fonctions prédéfinies
- **Ce cours se limitera à présenter un sous ensemble réduit des fonctionnalités au sein d'Excel**

Macro-commande ?

- **Automatisation d'actions répétitives**
- **Macros préprogrammées**
  - *Exemples :*
    - Powerpoint : ovale, cercle, rectangle, carré, segment...
    - Excel : fonctions.

Édition du code d'une macro

- **Accès**
  - Outils de développement
    - Options → personnaliser le Ruban → Ajout de l'onglet « Développeur »
    - Bouton « macros »
  - Raccourci Alt+F8.
- **Action**
  - Sélectionner la macro voulue,
  - Cliquer le bouton Modifier.

A faire sur votre PC

Création d'une macro

Classe	Nom	Note 1	Note 2	Note 3	Moyenne
Maximum		20	20	20	
Thomas	Guyet	12,00	8,00	11,00	10,33
Olivier	Dumin	15,00	14,00	17,00	15,33
Franck	Malleville	8,00	5,00	9,00	7,33
Armelle	Gbptel	12,00	13,00	15,00	13,33
Virginie	Cucunier	17,00	13,00	12,00	13,00
	Moyennes	12,80	10,00	12,80	11,87

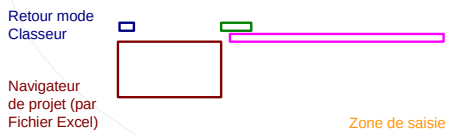
- **Enregistrement d'une macro sous Excel**
  - Ouvrir le fichier `ClasseurTravail_Base.xls` dans la feuille `base`
  - Placer votre curseur sur la case « Prénom » de la feuille Basse
    - Utilisation de « référence relatives » ??
  - Lancer l'enregistrement de votre macro
    - La nommer « comme vous voulez » puis validez ...
  - Effectuer les opérations suivantes
    - Calcul des moyennes par étudiants
    - Calcul de la moyenne de classe
    - La mise en forme : bordures + couleurs pour les moyennes

Exécution d'une macro

- **Exécution d'une macro**
  - Principe : Pour exécuter votre macro, ouvrez la fenêtre de macro **Alt+F8,** sélectionner votre macro puis cliquer sur **Exécuter**
  - Sur la feuille « `Test_Execution1` »
    - Placer le curseur sur la cellule « Prénom »
    - Lancer la macro
  - Sur la feuille « `Test_Execution2` »
    - Placer le curseur sur la cellule « Prénom »
    - Lancer la macro
    - Placer le curseur sur la cellule **B12**
    - Lancer la macro
  - Constater et analyser

À l'intérieur d'une macro ...

- **Avec l'icône macro lancer l'interface de modification d'une Macro**
  - **Alt+F8** puis sélectionner une macro et cliquer sur Editer
  - Touches de raccourcis : **alt+F11**



Exemple de macro enregistrée

```
Range("G3").Select
ActiveCell.FormulaR1C1 = "Moyenne"
Range("G5").Select
ActiveCell.FormulaR1C1 = _
    "=SUMPRODUCT(RC[-3]:RC[-1],R4C[-3]:R4C[-1])/SUM(R4C[-3]:R4C[-1])*20"
Range("G5").Select
ActiveCell.FormulaR1C1 = _
    "=SUMPRODUCT(RC[-3]:RC[-1],R4C[-3]:R4C[-1])/SUM(R4C[-3]:R4C[-1])"
Range("G5").Select
Selection.AutoFill Destination:=Range("G5:G9"), Type:=xlFillDefault
Range("G5:G9").Select
Range("D10").Select
ActiveCell.FormulaR1C1 = "=AVERAGE(R[-5]C:R[-1]C)"
Range("D10").Select
Selection.AutoFill Destination:=Range("D10:G10"), Type:=xlFillDefault
```

```
Range("D10:G10").Select
Range("G3:G9").Select
Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Borders(xlDiagonalUp).LineStyle = xlNone
With Selection.Borders(xlEdgeLeft)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlEdgeTop)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlEdgeRight)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
Range("D10:G10").Select
Selection.Font.Bold = True
Range("I9").Select
```


VBA un langage orienté objet

- **Les « classes »**
  - caractérisée par ses **propriétés**
  - permet de faire certaines actions : les **méthodes**
  - Exemple : Vehicule
    - Propriétés : Nombre de roue, Nombre de siège, Moteur (O/N)
    - Méthodes : rouler, …
- **Les « instances de classe » ou « objets »**
  - Une instance est la réalisation concrète d'une classe
  - Exemples :
    - Citroen C1 immatriculée 345 FI 456 : 4 roues, 5 sièges, O
    - Vélo immatriculé 34523456 : 2 roues, 2 sièges, N

VBA un langage orienté objet

- **En VBA tout est « objet »**
  - Je ne reviendrai pas trop sur les classes
  - Il faut surtout retenir que **les objets ont des propriétés/méthodes** et que en fonction de la nature des objets (leur classe), ces propriétés changent !
- **Une notation importante : la notation pointée**
  - Donne la propriété d'un objet
  - La valeur de la propriété peut être modifiée

```
Range("G3").Select
ActiveCell.FormulaR1C1 = "Moyenne"
Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Font.Bold = True
```


VBA un langage orienté objet

- **Notations pointées … simplifiées**
  - VBA utilise le contexte pour vous faciliter les écritures
    - **Range("G3").Select** est en fait
    - **ActiveWorkbook.ActiveSheet.Range("G3").Select**
  - La notation **With** permet de regrouper des préfixes

```
With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
```

```
Selection.Borders(xlEdgeBottom).LineStyle = xlContinuous
Selection.Borders(xlEdgeBottom).ColorIndex = 0
Selection.Borders(xlEdgeBottom).TintAndShade = 0
Selection.Borders(xlEdgeBottom).Weight = xlThin
```


Sources d'informations

- **Sur le langage et l'environnement**
  - Guide VBA (fonctions de base et propriétés les plus utiles)
  - Aide de l'éditeur VBA
  - Informations diverses sur l'internet
- **En cas de problèmes (80 % du tps à corriger les bugs!)**
  - Déjà, ***réfléchir***
    - Comparer code et résultat
    - Lire attentivement les messages d'erreur
  - ***Tester*** et ***re-tester*** avec de petites modifications
  - Rechercher le texte d'une erreur
    - Dans l'aide ou sur l'internet

Macros en VBA

- Forme générale
  - Des « procedures » structurés dans des « *Modules* »
  - Des commentaires
- Une macro est une procédure
  - Commence par « **Sub** *NomDeLaMacro()* »
  - Finit par « **End Sub** »
  - Contient une des instructions et des commentaires
- Une macro pourra être associée à des événements
  - Par ex : clic sur un bouton

Macros en VBA

```
' Commentaire
Sub NomMacro()
    ' Commentaire
   Instruction 1
   Instruction 2
   ...
   Instruction n
End Sub

' Blah blah
Sub NomMacro2()
    ' etc...
End Sub
```

- **Commentaires**
  - Texte libre
  - Destiné aux programmeurs
  - Ni trop, ni trop peu
    - « Ce que fait la macro »
    - Détails un peu compliqués
- **Procédures**
  - Contenu indenté : pour des question de lisibilité

Premier exemple

- **Modification de votre macro précédente**
  - Ouvrir votre macro en modification (**Alt+F11**)
    - Chercher votre macro dans le `Module 1`
  - Tester quelques modifications de votre macro
    - Changer les textes créés
    - Modifier les couleurs choisies
  - Ré-exécuter votre macro après modification (touche F5)

Plan du cours

- **Visual Basic for Applications**
- **Dans la macro il y a...**
- **Diviser pour régner**
- **Le problème derrière le clavier**

Dans la macro il y a ...

- **Des instructions**
  - Une **instruction** est un expression qui indique une action a réaliser par l'ordinateur
  - Exemples
    - Modification du contenu d'une variable et de sa mise en forme
    - Mais bien d'autres …
  - Remarque : une macro enregistrée est une succession d'instructions

Second exemple

- **Création d'une première macro**
  - Vous travaillez maintenant sur la feuille « `Tarifs` »
  - Création d'un module
  - Création d'une procédure
  - Écrire la procédure pour qu'elle
    - Donne une valeur à une cellule (par ex. la cellule **G5**)
    - Calculer le contenu d'une cellule en fonction des autres
      - Prix total pour un produit
  - Exécution d'un procédure
    - Par l'interface
    - Utilisation de la touche F5

Fonctions utiles :
- `Cells(row, column)`
- opérations arithmétiques

Second exemple

- **Programmer des formules Excel en VBA**
  - Il y a **deux niveaux de calculs**
    - Les formules du tableau Excel calcule des valeurs qui sont affichées dans une feuille Excel
    - VBA calcule des contenus de cellules du tableau Excel : valeur … ou formule
  - Une **formule Excel** est une *valeur* pour un programme VBA

    Cells(4,7) = **''coucou''**
    Cells(5,7)= **''=E5*D5+F5''**

    - Entre guillemets : comme le texte saisie …
  - Il faut savoir jouer avec ses 2 niveaux

Dans la macro il y a...

- **Variable ??**
  - « Boîte » contenant des données
    - Correspond à une position en mémoire
    - Possède un type
  - Accès au contenu de la boîte par une « étiquette »
    - En l'occurrence le nom de la variable
  - Permet de stocker provisoirement des valeurs
    - C'est une « mémoire » interne au programme
    - Possède un **type** qui indique *comment* on manipule le contenu
  - On doit d'abord *déclarer* la variable pour que le programme sache comment la manipuler

Dans la macro il y a...

- **« Déclarer » une variable**
  - Créer la « boîte » et lui mettre une « étiquette »,
  - Déclarer ce qu'on voudra mettre dans la « boîte »
    - Définition de la taille (espace mémoire à réserver),
    - Définition du type de codage utilisé.
      - Le type est en fait optionnel pour VBA … mais recommandé

```
Dim X As Integer
Dim tot As Double
Dim txt As String
X=5        'Affectation de la variable
Cells(X,7) = ''coucou''    ' Utilisation de la variable
```


Types de variables de base

- **Types numériques**
  - **Integer** : entier de -2 147 483 648 à 2 147 483 647,
    - Attention : ça « tourne »
      2147483647 + 1 = -2147483648
  - **Double** : réel de -1,79x10^{308} à 1,79x10^{308}
    - Attention : plus le nombre est gros, moins il est précis
- **Types alphanumériques**
  - **Char** : un caractère (lettre, chiffre, ponctuation...)
  - **String** : une suite de caractères.
- **Type booléen**
  - Deux valeurs possibles : vrai ou faux.
  - **Boolean**
    - **True** pour vrai
    - **False** pour faux

Dans la macro il y a...

- **Expressions**
  - Une série d'opérations qui donnent un résultat
  - Opérateurs
    - Par exemple arithmétique basique : +, -, *, /
    - Priorité des opérateurs
  - Valeurs, références à des variables
  - « Sous-expressions » entre parenthèses
    - Pour contourner les priorités (par exemple « (1 + 2) * 3 »)
    - Pour rendre le code plus lisibles, parfois

Dans la macro il y a...

- **Expressions**
  - « 1 » : la valeur 1
  - « 1 + 2 » : résultat de l'addition des valeurs 1 et 2
  - « x + 2 » : résultat de l'addition du contenu de la variable x à la valeur 2
  - « x * (2 + y) » : résultat de la multiplication du contenu de la variable x par la somme de la valeur 2 et du contenu de la variable y

Petit exemple

- **Illustration de l'utilisation des variables dans VBA**
  - Calcul des totaux
    - Calculer le total vendu sans visualiser le prix des ventes de chaque ligne !

Ici, on ne demande pas de mettre une formule Excel qui calcule le prix dans la cellule, mais de calculer en VBA une valeur qui sera donnée comme valeur de cellule.

Dans la macro il y a...

- **Instructions affectant l'application**
  - Exemples
    - Modification du contenue d'une variable
    - Modification du texte d'une cellule
    - Modification de la mise en forme d'une cellule
    - Mais bien d'autres ...
  - La forme peut paraître un peu obscure
    - Il y a beaucoup fonctions différentes (et on se limite à Excel)
    - Consulter les aides (documents fournis et doc en ligne)
    - Faites vous vos habitudes : pratiquez et vous retiendrez !!

Dans la macro il y a...

- **Instructions affectant l'application**
  - Modification du texte d'une cellule

```
Sub MacroPerso()
   Cells( 1 , 1 ) = "lol"
End Sub
```

  - Ajout de bordures à une cellule

```
Sub MacroPerso2()
   Cells( 1 , 1 ).Borders.LineStyle = xlContinuous
End Sub
```


Dans la macro il y a...

- **Structure arborescente**
  - **Cells(y,x)** « contient »
    - **Borders** qui « contient »
      - **LineStyle** qui indique le style de ligne
      - **ColorIndex** qui indique un numéro couleur

```
Sub MacroPerso2Bis()
   Cells( 1 , 1 ).Borders.LineStyle = xlContinuous
   Cells( 1 , 1 ).Borders.ColorIndex = 4
End Sub
```


Dans la macro il y a...

- **Structure arborescente**
  - Bloc **With** … **End With**
  - Deux lignes de plus, mais le contenu est beaucoup plus lisible

```
Sub MacroPerso2Ter()
   With Cells( 1 , 1 ).Borders
      .LineStyle = xlContinuous
      .ColorIndex = 4
   End With
End Sub
```


Cells et Range

- **Cells vs Range**
  - Deux fonctions pour sélectionner des cellules
    - Cells : utilise des indices numériques au sein d'une feuille
    - Range :
      - Utilise les nommages de cellules (comme dans les formules)
      - Permet de sélectionner des plages de cellules

```
Sub MacroPerso2Ter()
   With Range( 'C3:H8' ).Borders
      .LineStyle = xlContinuous
      .ColorIndex = 4
   End With
End Sub
```


Cells et Range

- ***Cells* vs *Range***
  - Deux fonctions pour sélectionner des cellules
    - Cells : utilise des indices numériques au sein d'une feuille
    - Range :
      - Utilise les nommages de cellules (comme dans les formules)
      - Permet de sélectionner des plages de cellules
- **Plus de détails sur la sélection de cellules dans Excel**
  - Accès aux autres feuilles voire à d'autres documents possible !
    - https://support.microsoft.com/fr-fr/help/291308/how-to-select-cells-ranges-by-using-visual-basic-procedures-in-excel

Variables et objets

- **Une variable peut « contenir » un objet constitutif du tableau Excel**
  - Range (ensemble de cellules), Sheet (feuille), Workbook (classeur)
- **L'affectation d'un tel objet se fait via l'instruction `Set`**

```
Dim Rng As Range
Set Rng = Range("H1:I45")

Dim S As Sheets
Set S = ActiveWorkbook.Sheets(1)
```


Macro comme **fonction** ... vers des fonctions personnalisées

- **Fonction**
  - Une fonction est une procédure qui **renvoie une valeur**
  - Du coup, une fonction a un type de retour
    - i.e. le type de la valeur que renvoie la fonction
- **Exemple**
  - « Je veux savoir quelle est le prix avec TVA si une cellule contient un certain nombre »
  - Paramètres :
    - La valeur d'une cellule ...
    - Type de TVA (5.5 vs 20 %)
  - Type de retour : valeur numérique

Macro comme **fonction** ... vers des fonctions personnalisées

- **Fonction**
  - Bloc **Function**...**End Function**
  - Indique le type de retour

```
Function TVA( prixht As Double, type55 As Boolean) As Double
    ' …


End Function
```


Macro comme **fonction** ... vers des fonctions personnalisées

- **Fonction**
  - Doit contenir un « assignement » au nom de la fonction elle-même
  - C'est ce qui détermine la *valeur renvoyée*

```
Function TVA( prixht As Double, type55 As Boolean) As Double
    ' …


   TVA = prixht * 1.205
End Function
```


Macro comme fonction ... vers des **fonctions personnalisées**

- **La fonction peut être utilisée comme une nouvelle fonction excel personnalisée**
- **À tester**
  - Écrire la fonction ci-dessous dans un Module de votre classeur et enregistrez
  - Revenir dans la feuille de calculs
  - Créer une formule de type « =TVA(C5) »

```
Function TVA( prixht As Double) As Double
   TVA = prixht * 1.205
End Function
```


Ce qu'on a vu ...

- Une Macro VBA manuelle est presque inutile
- Une macro est un programme informatique qui décrit des opérations sur le tableau Excel
  - Modification du contenu d'une cellule
  - Modification de la mise en forme d'une cellule
- Les éléments du tableau Excel sont accessibles en VBA comme des propriétés d'objets
  - Les cellules sont accessibles par l'expression `Cells(r,c)`
- Une macro VBA peut définir une formule du tableur
- Il est possible de créer des fonctions personnalisées

Dans la macro il y a...

- **Si on se limitait à ça...**
  - Seulement des tâches simples
  - Obligation de lister toutes les étapes manuellement
  - Aucune interaction avec le contenu du classeur
  - Aucune interaction avec l'utilisateur
  - Macros potentiellement énormes
- **On n'y gagnerait pas grand-chose**

Dans la macro il y a...

- **« Itération »**
  - Répéter une opération, à quelques détails près
  - Par exemple, « mettre une bordure autour des 100 premières cellules de la première colonne »

Dans la macro il y a...

- **« Itération »**

```
Sub MacroPerso3()
   Cells(4, 8) = Cells(4, 5) * Cells(4, 6)
   Cells(5, 8) = Cells(5, 5) * Cells(5, 6)
   Cells(6, 8) = Cells(6, 5) * Cells(6, 6)
   Cells(7, 8) = Cells(7, 5) * Cells(7, 6)
   Cells(8, 8) = Cells(8, 5) * Cells(8, 6)

   ' Ici, le programmeur s'est pendu
End Sub
```


Dans la macro il y a...

- **« Itération »**
  - On voudrait **répéter l'instruction** pour chaque ligne
  - Deux choses nécessaires
    - Le numéro de la ligne à modifier
    - Une structure indiquant la répétition

```
Sub MacroPerso3()
   ???    ' Faire répéter la ligne suivante
          ' en faisant changer le numéro de ligne de 1 à 100
   Cells(???, 8) = Cells(???, 5) * Cells(???, 6)
   ' Ici on aura besoin du numéro de ligne
End Sub
```


Dans la macro il y a...

- **« Itération »**
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - **Le numéro de la ligne à modifier**
    - Une structure indiquant la répétition

```
Sub MacroPerso3()
   ???    ' Faire répéter la ligne suivante
          ' en faisant changer ligne de 1 à 100
   Cells(ligne, 8) = Cells(ligne, 5) * Cells(ligne, 6)
   ' Mais d'où vient « ligne » ?
End Sub
```


Dans la macro il y a...

- **« Itération »**
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - **Le numéro de la ligne à modifier**
      - Change à chaque itération
        - → C'est une ***VARIABLE***
    - Une structure indiquant la répétition

Dans la macro il y a...

- **« Itération »**
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - **Le numéro de la ligne à modifier**
    - Une structure indiquant la répétition

```
Sub MacroPerso3()
   Dim ligne As Integer
   ???    ' Faire répéter la ligne suivante
          ' en faisant changer ligne de 1 à 100
   Cells(ligne, 8) = Cells(ligne, 5) * Cells(ligne, 6)
End Sub
```


Dans la macro il y a...

- **« Itération »**
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - Le numéro de la ligne à modifier
    - **Une structure indiquant la répétition**

```
Sub MacroPerso3()
   Dim ligne As Integer
          ' Faire répéter la ligne suivante
          ' en faisant changer ligne de 1 à 100
   Cells(ligne, 8) = Cells(ligne, 5) * Cells(ligne, 6)
End Sub
```


Dans la macro il y a...

- **« Itération »**
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - Le numéro de la ligne à modifier
    - **Une structure indiquant la répétition**
      - « Une boucle For »
      - Il s'agit d'un bloc
        - » Commence par **For**
        - » Finit par **Next**
      - Peut contenir n'importe quelle séquence d'instructions

Dans la macro il y a...

- « Itération »
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - Le numéro de la ligne à modifier
    - **Une structure indiquant la répétition**

```
Sub MacroPerso3()
   Dim ligne As Integer
   For ??? ' « ligne de 1 à 100 »
      Cells(ligne, 8) = Cells(ligne, 5) * Cells(ligne, 6)
   Next
End Sub
```


Dans la macro il y a...

- « Itération »
  - On voudrait répéter l'instruction pour chaque ligne
  - Deux choses nécessaires
    - Le numéro de la ligne à modifier
    - **Une structure indiquant la répétition**

```
Sub MacroPerso3()
   Dim ligne As Integer
   For ligne = 1 To 100
      Cells(ligne, 8) = Cells(ligne, 5) * Cells(ligne, 6)
   Next ligne
End Sub
```


Exemples pratiques

- **Traitement de toutes les lignes avec le calcul**
  - Quelles sont les bonnes limites ??
    - Faire attention à bien mettre ses limites
    - Mieux : calculer la limite à partir des données

```
Dim dl As Integer
dl = Range("H65536").End(xlUp).Row
```

  - Ajouter une ligne et re-tester
  - Pour information … dans les fonctions
    - Utilisation de la propriété count (cf TP)
- **Ajouter une colonne avec le calcul automatique d'un indice**

Dans la macro il y a...

- **Si on se limitait à ça...**
  - Seulement des tâches simples
  - ~~Obligation de lister toutes les étapes manuellement~~
  - Aucune interaction avec le contenu du classeur
  - Aucune interaction avec l'utilisateur
  - Macros potentiellement énormes
- **On y gagnerait juste un peu de temps**

Dans la macro il y a...

- **Exécution conditionnelle**
  - « Si *quelque chose* alors faire *un truc* sinon faire *un autre truc* »
  - Par exemple, « si le numéro de la ligne est pair, alors mettre une bordure noire, sinon mettre une bordure verte »

Dans la macro il y a...

- **Exécution conditionnelle**

```
Sub MacroPerso5()
    Dim ligne As Integer
    For ligne = 1 To 100
        Dim couleur As Integer
        ' Ici on détermine la couleur
        Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
        Cells( ligne , 1 ).Borders.ColorIndex = couleur
    Next
End Sub
```


Dans la macro il y a...

- **Exécution conditionnelle**
  - Blocs **If** … **Then** … **Else** … **End If**

```
Sub MacroPerso5()
    Dim ligne As Integer
    For ligne = 1 To 100
        Dim couleur As Integer
        If ??? Then
            ??? ' Mettre 1 (noir) dans couleur
        Else
            ??? ' Mettre 4 (vert) dans couleur
        End If
        Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
        Cells( ligne , 1 ).Borders.ColorIndex = couleur
    Next
End Sub
```


Dans la macro il y a...

- **Exécution conditionnelle**
  - *Assignement* : mettre une valeur dans une variable

```
Sub MacroPerso5()
    Dim ligne As Integer
    For ligne = 1 To 100
        Dim couleur As Integer
        If ??? Then
            couleur = 1 ' Noir
        Else
            couleur = 4 ' Vert
        End If
        Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
        Cells( ligne , 1 ).Borders.ColorIndex = couleur
    Next
End Sub
```


Dans la macro il y a...

- **Exécution conditionnelle**
  - Condition ?
    - Une expression dite *booléenne*
    - C'est-à-dire que son résultat est soit « vrai » soit « faux »
  - En l'occurrence on veut tester la parité du contenu de la variable *ligne*

```
        If ??? Then
            couleur = 1 ' Noir
        Else
            couleur = 4 ' Vert
        End If
```

Dans la macro il y a...

- **Exécution conditionnelle**
  - Tester la parité ?
    - Une valeur est paire si le *reste* de sa division entière par 2 est 0
    - Une valeur est impaire si le *reste* de sa division entière par 2 est 1
  - Reste de la division entière ?
    - Opérateur « mod »

```
If (ligne Mod 2) = 0 Then
   couleur = 1 ' Noir
Else
   couleur = 4 ' Vert
End If
```


Dans la macro il y a...

- **Exécution conditionnelle**
  - Tester l'égalité ?
    - Opérateurs de comparaison
      - =, <, > (égal, inférieur, supérieur)
      - <= et >= (inférieur ou égal, supérieur ou égal)
      - <> (différent)

```
If ligne Mod 2 = 0 Then
   couleur = 1 ' Noir
Else
   couleur = 4 ' Vert
End If
```


Dans la macro il y a...

- **Expressions booléennes plus complexes**
  - Exemples :
    - « Si *quelque chose* **et** *autre chose* alors... »
    - « Si *quelque chose* **ou** *autre chose* alors... »
    - « Si **pas** *quelque chose* alors... »
  - Opérateurs logiques
    - **And, Or** et **Not**, respectivement

Dans la macro il y a...

- **Expressions conditionnelles**
  - Parfois on ne veut faire quelque chose que dans un cas
    - « Si X alors … sinon RIEN. »

Dans la macro il y a...

- **Expressions conditionnelles**
  - Parfois on ne veut faire quelque chose que dans un cas
    - « Si X alors … sinon RIEN. »
    - On pourrait faire quelque chose comme ça...

```
If ligne Mod 2 = 0 Then
   Cells( ligne , 1 ) = "Gné !"
Else
   ' Rien.
End If
```


Dans la macro il y a...

- **Expressions conditionnelles**
  - Parfois on ne veut faire quelque chose que dans un cas
    - « Si X alors … sinon RIEN. »
    - Donc on fait comme ça :

```
If ligne Mod 2 = 0 Then
   Cells( ligne , 1 ) = "Gné !"
End If
```

Dans la macro il y a...

- **Expressions conditionnelles**
  - Conditions multiples
    - « Si X alors … sinon si Y alors … sinon si Z alors … »
    - On pourrait imbriquer...

```
If ligne Mod 3 = 0 Then
   Cells( ligne , 1 ) = "Gné !"
Else
   If ligne Mod 3 = 1 Then
      Cells( ligne , 1 ) = "Blah !"
   Else
      Cells( ligne , 1 ) = "Haha !"
   End If
End If
```


Dans la macro il y a...

- **Expressions conditionnelles**
  - Conditions multiples
    - « Si X alors … sinon si Y alors … sinon si Z alors … »
    - Mieux … mais attention à l'ordre des tests

```
If ligne Mod 3 = 0 Then
   Cells( ligne , 1 ) = "Gné !"
ElseIf ligne Mod 3 = 1 Then
   Cells( ligne , 1 ) = "Blah !"
Else
   Cells( ligne , 1 ) = "Haha !"
End If
```


Exemple

- **Exemple pratique**
  - Mettre en rouge la cellule de commission lorsque que la commission totale sera inférieure à 40€, sinon mettre la cellule en vert
  - Infos utiles
    - La marge est données par `prix*quantité*commission/100`
    - Modification de la couleur de fond : `.Interior.ColorIndex`

Dans la macro il y a....

- **Itérations, bis**
  - Boucle **For** vue plus tôt très limitée
    - Permet simplement de « compter » entre deux valeurs
  - Parfois on veut faire quelque chose...
    - … *tant qu'*une condition n'est pas atteinte
    - … *jusqu'à ce qu'*une condition soit atteinte
  - L'un et l'autre sont équivalents
    - « Jusqu'à ce que X » → « Tant que pas X »

Dans la macro il y a....

- **Itérations, bis**
  - Exemple : « Mettre une bordure autour des 100 premières cellules de la première colonne, sauf si une cellule contient le mot FIN auquel cas on s'arrête ».

```
Sub MacroPerso6()
   Dim ligne As Integer
   For ligne = 1 To 100
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ' Comment faire pour ne pas continuer ?
   Next
End Sub
```


Dans la macro il y a....

- **Itération, bis**
  - Boucle « **While** »

```
While ??? ' Condition
   ' Actions
Wend
```

  - Boucle « **Do While** »

```
Do
   ' Actions
Loop While ??? ' Condition
```

  - Différence ?
    - **Do**...**Loop While** s'exécute toujours une fois

Dans la macro il y a....

- **Itérations, bis**
  - On va donc utiliser une boucle **While**
    - Mais il va falloir compter « à la main »
  - Commençons par là.
    - Ci-dessous, boucle équivalente à la précédente.

```
Sub MacroPerso6()
   Dim ligne As Integer
   ligne = 1
   While ligne <= 100
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ligne = ligne + 1
   Wend
End Sub
```


Dans la macro il y a....

- **Itérations, bis**
  - On veut aussi regarder la cellule actuelle
    - Tant qu'on n'est pas à la ligne 100 **et** que la cellule ne contient pas « FIN »

```
Sub MacroPerso6()
   Dim ligne As Integer
   ligne = 1
   While ligne <= 100 And Cells( ligne , 1 ).value <> "FIN"
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ligne = ligne + 1
   Wend
End Sub
```


Dans la macro il y a....

- **Itérations, bis**
  - Hmmm… Là, il y a un petit bug...

```
Sub MacroPerso6()
   Dim ligne As Integer
   ligne = 1
   While ligne <= 100 And Cells( ligne , 1 ).value <> "FIN"
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ligne = ligne + 1
   Wend
End Sub
```


Dans la macro il y a....

- **Itérations, bis**
  - Hmmm… Là, il y a un petit bug…
    - Quand on arrive à la ligne 101, on la teste quand même !
    - Pas un problème dans ce cas précis, mais cela pourrait.

```
Sub MacroPerso6()
   Dim ligne As Integer
   ligne = 1
   While ligne <= 100 And Cells( ligne , 1 ).value <> "FIN"
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ligne = ligne + 1
   Wend
End Sub
```


Dans la macro il y a....

- **Itérations, bis**
  - De manière générale
    - Se méfier des conditions complexes
    - Attention aux boucles infinies !

```
Sub MacroQuiPlante()
   Dim ligne As Integer
   ligne = 1
   While ligne <= 100 And Cells( ligne , 1 ).value <> "FIN"
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
   Wend
End Sub
```


Dans la macro il y a...

- **Si on se limitait à ça...**
  - ~~Seulement des tâches simples~~
  - ~~Obligation de lister toutes les étapes manuellement~~
  - ~~Aucune interaction avec le contenu du classeur~~
  - Aucune interaction directe avec l'utilisateur
  - Macros potentiellement énormes

Plan du cours

- **Qu'est-ce que la programmation ?**
- **Les langages de programmation**
- **Visual Basic for Applications**
- **Dans la macro il y a...**
- **Diviser pour régner**
- **Le problème derrière le clavier**

Diviser pour régner

- **Imaginons que l'on veuille...**
  - Colorier en vert toutes les cellules des 100 premières lignes de la colonne A contenant le mot « VERT »
  - Colorier en rouge toutes les cellules des 200 premières lignes de la colonne B contenant le mot « ROUGE »
  - Colorier en bleu toutes les cellules des 150 premières lignes de la colonne C contenant le mot « BLEU »

Diviser pour régner

- **Et de un...**

```
Sub MacroIdiote7()
   Dim ligne As Integer
   For ligne = 1 To 100
      If Cells( ligne , 1 ).value = "VERT" Then
         Cells( ligne , 1 ).Interior.ColorIndex = 4
      End If
   Next
End Sub
```


Diviser pour régner

- **Et de deux...**

```
Sub MacroIdiote7()
   Dim ligne As Integer
   For ligne = 1 To 100
      If Cells( ligne , 1 ).value = "VERT" Then
         Cells( ligne , 1 ).Interior.ColorIndex = 4
      End If
   Next
   For ligne = 1 To 200
      If Cells( ligne , 2 ).value = "ROUGE" Then
         Cells( ligne , 2 ).Interior.ColorIndex = 3
      End If
   Next
End Sub
```


Diviser pour régner

```
Sub MacroIdiote7()
   Dim ligne As Integer
   For ligne = 1 To 100
      If Cells( ligne , 1 ).value = "VERT" Then
         Cells( ligne , 1 ).Interior.ColorIndex = 4
      End If
   Next
   For ligne = 1 To 200
      If Cells( ligne , 2 ).value = "ROUGE" Then
         Cells( ligne , 2 ).Interior.ColorIndex = 3
      End If
   Next
   For ligne = 1 To 150
      If Cells( ligne , 3 ).value = "BLEU" Then
         Cells( ligne , 3 ).Interior.ColorIndex = 5
      End If
   Next
End Sub
```


Diviser pour régner

- **Problèmes de l'approche « Copy/paste »**
  - Ça devient long et illisible
  - S'il y a un bug dans le code, il faudra corriger partout
  - Facile d'oublier de modifier un détail
- **Solution**
  - Identifier les points communs
  - Identifier ce qui change
  - Extraire et séparer le code correspondant

Diviser pour régner

- **Pour en revenir à l'exemple...**

```
Dim ligne As Integer
For ligne = 1 To 100
   If Cells( ligne , 1 ).value = "VERT" Then
      Cells( ligne , 1 ).Interior.ColorIndex = 4
   End If
Next
```


Diviser pour régner

- **Pour en revenir à l'exemple...**

```
Dim ligne As Integer
For ligne = 1 To 200
   If Cells( ligne , 2 ).value = "ROUGE" Then
      Cells( ligne , 2 ).Interior.ColorIndex = 3
   End If
Next
```


Diviser pour régner

- **Pour en revenir à l'exemple...**

```
Dim ligne As Integer
For ligne = 1 To 150
   If Cells( ligne , 3 ).value = "BLEU" Then
      Cells( ligne , 3 ).Interior.ColorIndex = 5
   End If
Next
```


Diviser pour régner

- **Utiliser des variables serait insuffisant**
  - Il faudrait tout de même copier/coller le code de la boucle

```
Dim ligne , nLignes , couleur , colonne As Integer
Dim texte As String
nLignes = 100
couleur = 4
texte = "VERT"
colonne = 1
For ligne = 1 To nLignes
   If Cells( ligne , colonne ).value = texte Then
      Cells( ligne , colonne ).Interior.ColorIndex = couleur
   End If
Next
' etc...
```


Diviser pour régner

- **Procédures paramétrées**
  - Une procédure, c'est une série d'actions
    - Une procédure peut être appelée par une autre procédure
    - Techniquement, une macro est une procédure
  - Une procédure, c'est un bloc **Sub**...**End Sub**
  - Une procédure paramétrée a des, heu, paramètres
    - Valeurs passées à la procédure lors de son appel
    - Se comportent (presque) comme des variables

```
Sub MaProcedure( parametre1 As type , parametre2 As type ... )
   ' Du code !
End Sub
```


Diviser pour régner

- **L'exemple, de nouveau**
  - On extrait la boucle dans une procédure paramétrée
  - La procédure a besoin du nombre de lignes, du texte à trouver, de la colonne et de la couleur

```
Sub ColoriageTexte( colonne As Integer ,
                    nLignes As Integer ,
                    couleur As Integer ,
                    texte As String )
   For ligne = 1 To nLignes
      If Cells( ligne , colonne ).value = texte Then
         Cells( ligne , colonne ).Interior.ColorIndex = couleur
      End If
   Next
End Sub
```

Diviser pour régner

- **L'exemple, de nouveau**
  - Dans la macro, on se contente d'appeler la procédure 3 fois, avec 3 jeux de paramètres

```
Sub MacroIdiote7()
   Call ColoriageTexte( 1 , 100 , 4 , "VERT" )
   Call ColoriageTexte( 2 , 200 , 3 , "ROUGE" )
   Call ColoriageTexte( 3 , 150 , 5 , "BLEU" )
End Sub
```


Diviser pour régner

- **Procédures parfois insuffisantes**
  - Pas de communication depuis la procédure appelée vers la procédure appelante
  - Problème si le but de la procédure est par exemple de calculer quelque chose

Diviser pour régner

- **Fonction**
  - Une fonction est une procédure qui renvoie une valeur
  - Du coup, une fonction a un type de retour
    - i.e. le type de la valeur que renvoie la fonction
- **Exemple**
  - « Je veux savoir si une cellule contient un certain texte »
  - Paramètres :
    - Coordonnées de la cellule
    - Texte à examiner
  - Type de retour : booléen
    - « oui elle contient ça » ou « non elle ne contient pas ça »

Diviser pour régner

- **Fonction**
  - Bloc **Function**...**End Function**
  - Indique le type de retour

```
Function ContientTexte( ligne As Integer ,
                        colonne As Integer ,
                        texte As String ) As Boolean
   ' ...
End Function
```


Diviser pour régner

- **Fonction**
  - Doit contenir un « assignement » au nom de la fonction elle-même
  - C'est ce qui détermine la *valeur renvoyée*

```
Function ContientTexte( ligne As Integer ,
                        colonne As Integer ,
                        texte As String ) As String
   ContientTexte = ( Cells( ligne, colonne ).Value = texte )
End Function
```


Diviser pour régner

- **Fonction**
  - Utilisation dans une expression
    - Par exemple assignement à une variable

```
Dim tutu As Boolean
tutu = ContientTexte( 1 , 1 , "TUTU" )
```

    - Ou dans une instruction conditionnelle

```
Sub MacroIdiote6Bis()
   Dim ligne As Integer
   ligne = 1
   While … Not ContientTexte( ligne , 1 , "TUTU" )
      Cells( ligne , 1 ).Borders.LineStyle = xlContinuous
      ligne = ligne + 1
   Wend
End Sub
```

Diviser pour régner

- **Si on se limitait à ça...**
  - ~~Seulement des tâches simples~~
  - ~~Obligation de lister toutes les étapes manuellement~~
  - ~~Aucune interaction avec le contenu du classeur~~
  - Aucune interaction directe avec l'utilisateur
  - ~~Macros potentiellement énormes~~

Plan du cours

- **Qu'est-ce que la programmation ?**
- **Les langages de programmation**
- **Visual Basic for Applications**
- **Dans la macro il y a...**
- **Diviser pour régner**
- **Le problème derrière le clavier**

Le problème derrière le clavier

- **« Parler » à l'utilisateur**
  - On pourrait écrire dans le classeur Excel, par exemple
    - Difficile à voir pour lui
    - S'il y a plusieurs messages ?
  - On utilise des boîtes de dialogue
    - La plus élémentaire : la boîte de message

```
Sub Yo()
    MsgBox( "Bonjour" )
End Sub
```

Le problème derrière le clavier

- **Demander son avis à l'utilisateur**
  - En lui proposant des choix précis
    - → Boîte de message avec boutons personnalisés

```
Sub PoserUneQuestionStupide()
    Dim reponse As Integer
    reponse = MsgBox( "Voulez-vous continuer ?" , vbYesNo )
    If reponse = vbYes Then
        MsgBox( "Z'êtes bien urbain." )
    End If
End Sub
```


Le problème derrière le clavier

- **Permettre à l'utilisateur de saisir du texte**
  - Boîte de saisie

```
Sub DetecteurDHorreursCosmiques()
    Dim reponse As String
    reponse = InputBox( "Entrez votre nom :" )
    If reponse = "Cthulhu" Then
        MsgBox( "Iä !" )
    End If
End Sub
```


Le problème derrière le clavier

- **Événements**
  - Certaines actions de l'utilisateur sont détectées par Excel
  - Excel essaie alors d'appeler certaines procédures dans le code VBA associé au classeur
  - Actions détectées ?
    - Sélection d'une plage de cellules
    - Saisie dans des cellules
    - Clic sur des boutons, etc
  - L'action détermine le nom de la procédure appelée

Le problème derrière le clavier

- **Quelques exemples pratiques**
  - Ajouter une question à l'utilisateur pour lui demander un numéro de couleur (indexée) à mettre sur les cellules dont la commission est <40€
  - Ajout d'un bouton pour déclencher votre commande

Quelques notions importante pour VBA sous Excel

- **Cellules actives ou sélectionnées**
  - Sélectionnées : sur fond bleu
  - Actives : bordures surlignées en gras
  - Elle peuvent être définies et mobilisées facilement
    - Fonctions **.Select** / **Selection.**
    - Fonctions **.Activate** / **ActiveCell.**

Environnement VBA

Objectifs (savoir-faire en VBA) :

- Savoir accéder à l'environnement de programmation VBA sous Excel
- Savoir créer un module, une procédure (macro) et exécuter la macro

1 *Configuration d'Excel pour le développement VBA*

Tout d'abord, nous allons configurer Excel pour la suite du TP. Par défaut, l'interface d'Excel ne donne pas accès aux fonctionnalités avancées que nous allons voir dans ce TP. Nous allons donc configurer le ruban (barre des icônes) pour qu'ils répondent à nos besoins.

1. Faire un clic-droit n'importe où sur le ruban, puis utiliser le menu **`Personnaliser le ruban`**

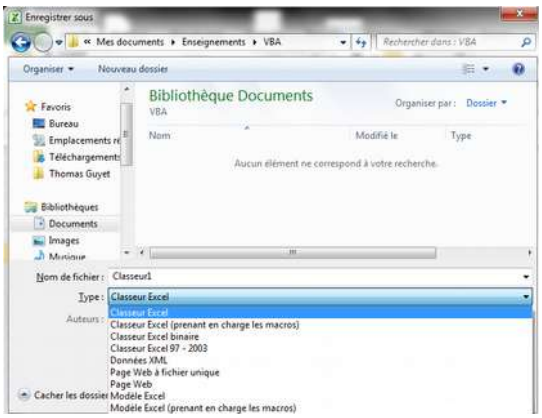
Il y a alors deux listes de fonctions : celle de gauche correspond à l'ensemble des fonctionnalités possibles dans Excel (structurée en catégories), celle de droite correspond à l'organisation du ruban sous forme hiérarchique (ruban, groupe et icônes).

2. Dans la liste de droite, cocher la case pour l'onglet du ruban « `Développeur` » pour activer l'onglet de ruban pour les développeur VBA
3. Valider avec `ok`

Vous devriez maintenant avoir accès au nouveau ruban pour les fonctionnalités de programmation VBA.

Dans ce TP nous allons enrichir le fichier *Excel* par des macros (fonctions VBA). Pour que ces macros soient enregistrées avec le fichier, il faut choisir le format de fichier `.xlsm` ! Le format de fichier `.xls` ou `.xlsx` ne conservera pas les macros !

4. Enregistrer votre fichier avec le format `xlsm`
   1. Dans le menu **`Fichier>Enregistrer sous...`**
   2. Choisir le type de fichier « `Classeur Excel (prenant en charge les macros)` » (*cf.* illustration ci-contre)



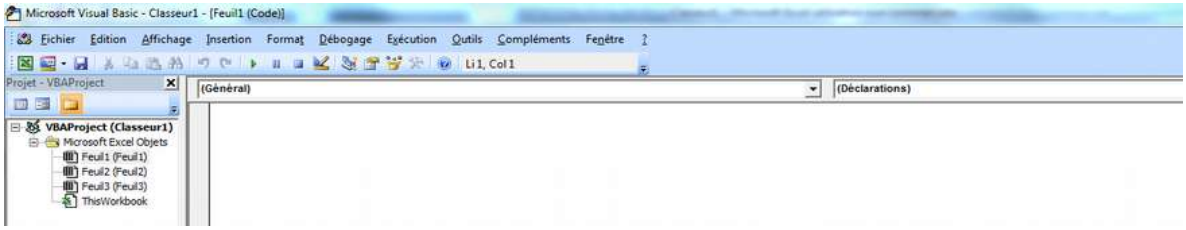
Ces configurations élémentaires ayant été faites, nous pouvons nous lancer dans la réalisation de macros VBA.

2 *Une première macro*

5. Depuis le ruban Développeur, utiliser l'icône Visual Basic pour lancer l'environnement de développement Visual Basic. Vous pouvez également utiliser la combinaison de touches **`Alt+F11`**.

Une fenêtre apparaît, dans laquelle on peut écrire des « macros » ou des « procédures » qui pourront être ensuite exécutées à l'aide de la touche **`F5`** dans un premier temps, ou en utilisant le menu **`Exécuter>Exécuter`**

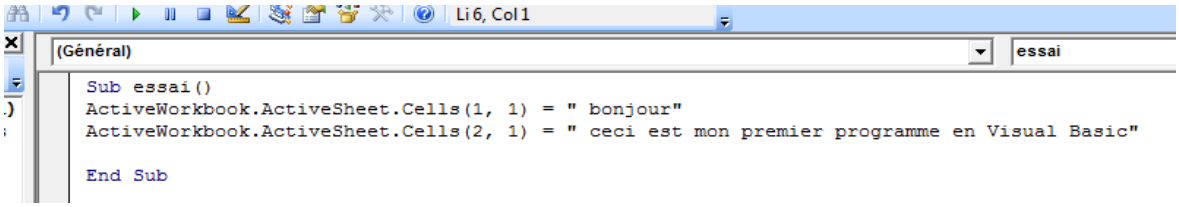
SubForm.



6. Créer un nouveau module de macros à partir du menu **Insertion>Module**
7. Renommer ensuite ce module : sélectionner le module dans la liste des modules de votre projet VBA puis, dans les propriétés, changer son nom

Nous allons créer une première macro pour tester le fonctionnement de l'environnement de programmation VBA. Une macro est une procédure introduite avec le mot clé `Sub`. Elle comporte une suite d'instructions qui vont s'exécuter les unes après les autres. La première ligne indique le nom de la macro et (éventuellement) les variables qu'elle doit traiter entre les parenthèses. La dernière ligne est toujours : `END SUB`. Elle indique la fin de la procédure.

8. Saisir la macro ci-dessous :



Quelques remarques sur l'éditeur VBA :

- Les programmes VBA sont insensibles à la casse … plus précisément, vous n'avez pas à faire attention aux majuscules lorsque vous tapez vos commandes, l'éditeur corrigera automatiquement les majuscules
- L'éditeur effectue une coloration syntaxique : les mots en vert sont des commentaires du programme, les mots en bleu sont des mots clés du langage
- Lorsque vous taper une apostrophe le texte qui suit n'est pas considéré comme une instruction exécutable, mais comme un commentaire. Cela permet de préciser certaines informations utiles au programmeur
- L'éditeur effectue de l'auto-complétion lorsqu'il reconnaît ce que vous voulez écrire, il peut proposer la fin du nom de votre commande. Il peut également ajouter automatiquement les fins de blocs de structure de contrôle

Cette macro sera incluse dans la liste des macros de votre module, et vous pourrez en demander l'exécution. Cette macro sera liée au classeur en cours, mais son exécution pourra concerner des données contenues dans d'autres classeurs.

9. Utiliser la touche **F5** pour exécuter votre macro. Si votre curseur se trouve dans une macro, alors vous exécutez celle-ci. En revanche, si votre curseur est en dehors de toute macro, une interface va s'ouvrir pour vous demander de choisir la macro à exécuter.

Fonctions personnalisées

Excel contient plus de 300 fonctions prédéfinies dans une feuille de travail. Mais si cela ne suffit pas, grâce à VBA, on peut créer des fonctions personnalisées. En effet, les fonctions personnalisées sont utiles dans les formules de feuille de travail Excel et les procédures VBA. Les fonctions personnalisées simplifient le travail, permettent à l'utilisateur de gagner du temps et permettent de raccourcir les formules de manière significative.

Objectifs (savoir-faire en VBA, par une procédure) :

- Définition et utilisation de fonctions personnalisées dans Excel
- Utiliser les paramètres d'une fonction (paramètre numériques et paramètres de type `Range`)
- Utilisation des structures de contrôles alternatives et itératives dans des fonctions

1 Première fonction

L'objectif du deuxième exercice est d'abord d'employer une fonction standard (incluse dans Excel par Microsoft) et non paramétrée (n'employant pas de donnée en provenance du classeur Excel), puis d'écrire une fonction personnalisée (construite par vous) et non paramétrée.

1. Ouvrir le fichier `FonctionsPerso.xls` et activer l'éditeur Visual Basic (**`Alt+F11`**)
2. Créer la fonction personnalisée (VBA) non paramétrée, qui permet de calculer plusieurs nombres tirés au hasard et compris entre 1 et 100.
   1. Créer un nouveau module pour votre fonction
   2. Ajouter la commande `Option Explicit`[1] en haut du module (en dehors des fonctions)
   3. Écrire, dans le module, la fonction suivante (sans les commentaires)

```
Function Hasard() As Byte
  Dim Aleatoire As Single
  Randomize (Now())
  Aleatoire = Rnd()
  Aleatoire = (Aleatoire * 99) + 1
  Hasard = Int(Aleatoire)
End Function
```

`Randomize (Nombre)` est une instruction qui initialise le générateur de nombres aléatoires. Pour que le générateur n'affiche pas la même série de valeurs lors de deux utilisations successives du programme, le paramètre de l'instruction doit être différent. Les informaticiens ont pris l'habitude d'insérer comme paramètre la fonction `Now` qui donne la date et l'heure en cours

`Rnd()` est une fonction qui mémorise un nombre aléatoire réel compris entre 0 et 1 proposé par le système

`Int(Nombre)` est une fonction qui renvoie la partie entière d'un nombre réel.

3. Enregistrer le contenu du module (**`Ctrl+S`**). Excel envoie un message qui vous indique que le format et l'extension du fichier (`.xlsx`) ne permettent pas d'enregistrer un programme écrit en VBA.
   Cliquer sur le bouton Non et choisir le format d'enregistrement « `Classeur Excel (prenant en charge les macros` », dont l'extension est `xlsm`
4. Insérer la fonction personnalisée dans la cellule `C5` d'une feuille de calcul et tester.
5. Recopier ensuite la fonction dans la place `C5:G5`

1L'option `Explicit` impose au programmeur de déclarer toutes les variables de son programme. Ceci force à faire des programmes plus lisibles et améliore les performances en temps et mémoire de la macro.

2 *Écriture d'une fonction paramétrée*

Une fonction paramétrée permet de calculer une valeur à partir de données stockées dans une feuille de calcul. La syntaxe d'une fonction paramétrée diffère de celle d'une fonction non paramétrée par son en-tête (sa première ligne). Il faut déclarer entre les parenthèses chaque paramètre (chaque information en provenance d'Excel et nécessaire au calcul). Pour chaque paramètre, doivent être indiqués le nom que vous lui donnez et son type.

A Exercice guidé

6. Ouvrir la feuille `Tuyaux`
7. Dans un module, écrire la fonction paramétrée `Section` tel que proposé ci-dessous.

```
Function Section(Diametre As Single) As Single
  Const PI = 3.14159
  Dim Rayon As Single
  Rayon = Diametre/2
  Section = Pi * Rayon * Rayon
End Function
```

Sachant le diamètre de chaque tuyau, la formule qui permet de calculer sa section est π * Rayon * Rayon. Le paramètre de la fonction (la donnée stockée dans la feuille Excel à partir de la laquelle on peut calculer la section) est donc Diamètre, une variable de type Single (réel simple). La fonction « Section » est de type Single, car la valeur de la section d'un tuyau est un nombre réel. Le calcul nécessite de connaître la valeur de π. Il faut déclarer la valeur de π comme une constante locale, c'est-à-dire une constante que vous déclarez pour le temps que dure la fonction. Le calcul nécessite de connaître la valeur du rayon. Il faut déclarer une variable locale « Rayon » de type Single et dont la valeur est égale à la moitié du diamètre.

8. Insérer la nouvelle fonction dans la cellule `D2` de la feuille de calcul (soit en écrivant dans la barre de formule, soit avec l'assistant de formule), puis recopier la fonction insérée en `D2` vers le bas, sur la plage `D2:D69`. Vous utiliserez le diamètre du tuyau, situé dans la colonne `A`.

Paramètre réel ?? On pourra noter que lors de l'utilisation de la fonction dans une formule dans le tableur, on utilise un nom de cellule comme paramètre de la fonction … et non un nombre. En fait, l'expression qui est contenu dans l'emplacement du paramètre de la fonction est évalué avant d'exécuter la fonction elle même.

On prend maintenant un cas un peu plus complexe … on veut mettre un calcul comme paramètre de la fonction … c'est tout à fait possible.

9. Dans la colonne `E`, appliquer de nouveau votre fonction, mais en utilisant le rayon du tuyau (colonne `B`). Vous devez recalculer le diamètre du tuyau en mm à partir de son rayon en cm pour pouvoir utiliser la fonction de manière adéquate : vous avez donc dans ce cas, *diametre=20*rayon*.

> Attention : si le type ou la classe de l'information calculée n'est pas correct, la fonction ne peut pas fonctionner et vous avez alors une erreur. C'est ce qui devrait s'être passé sur la dernière ligne de votre tableau.

B À vous de jouer ...

Sachant que le volume d'un cylindre est égal à Volume = π * Rayon * Rayon * Hauteur , créer à la suite de la fonction `Section`, une fonction que vous nommerez `Cylindre`. Cette fonction nécessite deux paramètres de type réel: « `Hauteur` » et « `Diamètre` ».

```
Function Cylindre(Diametre As Single, Hauteur As Single) As Single
```

Vous testerez ensuite votre fonction dans la colonne `D` de la feuille de calculs `Tuyau`.

3 *Fonctions avec cellule(s) en paramètre*

Dans l'exemple précédent, on a vu que les fonctions pouvaient prendre en compte des paramètres qui sont des nombres, mais si on veut faire une fonction dépend d'une propriété d'une cellule (par exemple, sa couleur), alors il faut être en mesure d'accéder aux propriétés de la cellule. L'objet qui doit être passé en paramètre sera alors une cellule et non une valeur.

> On peut noter qu'une limite d'une fonction personnalisée VBA est l'**impossibilité de modifier les propriétés de la cellule** (autre que sa valeur). On ne peut pas faire une fonction qui va faire du formatage conditionnel. Il faut alors recourir plutôt à une procédure qui traitera des cellules en paramètre et qui sera appelée lors de la mise à jour de la feuille. Cela dépasse le contenu de ce TP.

L'objectif ici est de créer une fonction capable de donner l'index de la couleur d'une cellule `=INDEX_COULEUR(C2)`. Bon, ça sert pas à grand-chose … mais on attend de savoir programmer un peu plus pour avoir des choses intéressantes !!

10. Implémenter la fonction ci-dessous dans votre module et testez la dans la feuille Couleur du tableau Excel
11. Ajouter dans une cellule la formule « =INDEX_COULEUR(23) »

```
Function INDEX_COULEUR(Cellule As Range) As Single
  INDEX_COULEUR = Cellule.Interior.ColorIndex
End Function
```

Ce qu'il faut noter sur cette fonction, c'est l'utilisation d'une cellule (objet de type Range) comme paramètre de la fonction lorsqu'on veut récupérer les propriétés de cette cellule.

4 *Insertion de fonctions dans des fonctions*

Il est possible d'insérer une fonction personnalisée dans une autre fonction personnalisée (écrite pour le même projet), comme il est possible d'insérer une fonction standard dans une fonction personnalisée. Il faut simplement veiller à ce que les paramètres de la fonction qui est appelée au sein de la fonction soient aussi déclarés comme paramètres dans la fonction qui appelle.

Le volume d'un cylindre est égal à *Volume = Section du cylindre * Hauteur*. Il est donc possible d'employer, dans la fonction `Cylindre`, la fonction `Section` pourvu que le paramètre *Diamètre* de la fonction Section soit aussi un paramètre de la fonction `Cylindre`.

En dessous de la fonction `Cylindre` que vous avez construite dans l'exercice précédent, écrire la fonction suivante :

```
Function Cylindre_2(Diametre As Single, Hauteur As Single) As Single
  Dim HauteurEnMm As Single
  HauteurEnMm = Hauteur * 1000
  Cylindre_2 = Section(Diametre) * HauteurEnMm
EndFunction
```

12. Implémenter la fonction ci-dessus dans votre module et tester la dans la colonne `F` de la feuille `Tuyau`.

5 Programmer des fonctions personnalisées : structures de contrôle

A Structure de contrôle alternative

A.1 Exemple

Dans la feuille Alternative, on donne quelques notes d'étudiants. On souhaite ajouter une information sur la validité d'une note (doit être comprise entre 0 et 20), sinon on affiche un message dans la case attenante.

On donne dans la colonne B le calcul réalisée avec une formule Excel :

```
=SI(ET(A5>=0;A5<=20); 'Note OK'; 'Note impossible')
```

Pour réaliser la même chose à l'aide d'une fonction VBA, vous devrez créer la fonction ci-dessous dans votre module qui réalisera la même opération

```
Function NoteValide(Note As Single) As String
 If (Note<=20) And (Note>=0) Then
       NoteValide = 'Note OK'
 Else
       NoteValide = 'Note impossible'
 End If
End Function
```

13. Utiliser ensuite votre fonction dans la colonne C de la feuille Alternative
14. Proposer une transformation de cette fonction pour utiliser un Or à la place d'un And dans le test

A.2 À vous de jouer

Une Faculté propose trois enseignements en première année de licence : économie, mathématiques et statistiques. Les coefficients appliqués à ses enseignements sont, respectivement, égaux à 3 (`PondEco`), 2 (`PondMath`) et 2 (`PondStat`). Entre parenthèses, ce sont des noms données aux cellules. À partir de ces données et d'un échantillon de notes, la Faculté veut tester plusieurs règlements d'examen.

- Le premier règlement stipule que pour réussir l'étudiant doit obtenir une moyenne générale supérieure ou égale à 10.
- Le deuxième règlement stipule que l'étudiant, pour réussir, doit obtenir une moyenne générale supérieure ou égale à 10 et une note en économie supérieure ou égale à 8.
- Le troisième règlement stipule que l'étudiant doit obtenir une moyenne générale supérieure ou égale à 10, une note en économie supérieure ou égale à 8, une note en matières quantitatives (moyenne pondérée des maths et des stats) supérieure ou égale à 8.

L'objectif est de compléter le tableau de la feuille `Alternative` pour savoir quelles règles permet à quels étudiants d'avoir le diplôme. La feuille comprend déjà le calcul des moyennes pondérées (en formule) dans la colonne E.

15. Commencer par donner des noms aux cellules pour la pondération (ce sera plus simple que l'utilisation de référence de cellules)
16. Réaliser une fonction VBA pour traduire le premier règlement (resp. second règlement) et appliquer cette fonction dans la colonne `F` (resp. `G`)
17. Créer une fonction qui réalise le calcule de la moyenne pondérée des matières quantitatives (NB : les noms de cellule peuvent être utilisés dans la fonction comme des variables)
18. Réaliser une fonction VBA pour traduire le troisième règlement et appliquer cette fonction dans la colonne `H`. Cette fonction fera appel à la fonction de la question précédente.

Pour faciliter le choix du règlement, on souhaite pouvoir choisir le règlement en donnant la valeur de la cellule `J10` (actuellement 2), et que en fonction de la valeur indiquée (le numéro du règlement), alors s'affiche dans la colonne `I` . Donc, en fonction de la valeur de `J10`, il faut *réaliser le bon calcul* de l'admissibilité.

19. Réaliser une fonction VBA qui réalise ce comportement (sans faire appel aux cellules des colonnes `F, G` ou `H`).

Bien sûr, il serait possible de peaufiner cette feuille en ajoutant un domaine de validité à la cellule `J10`, en ajoutant du formatage conditionnel, etc. Ce n'est pas notre but ici … je vous renvois aux cours de l'année passée.

B Structure de contrôle itérative

B.1 Exemple

Dans la feuille `Iterations`, vous trouverez toujours une liste de notes d'étudiants. Cette fois, nous allons nous intéresser à réaliser une fonction du calcul de la moyenne en VBA (oui ! Ca existe déjà, mais on va l'ajuster à notre besoin dans un second temps).

L'objectif est donc de réaliser une fonction VBA qui permette de calculer la moyenne de ET1 à l'aide d'une formule de type :

```
=MOYENNE_PERSO(B5:D5)
```

La particularité étant l'utilisation d'un range de cellule comme paramètre de la fonction (on a vu l'utilisation pour une cellule, on le regarde ici pour plusieurs cellules).

```
Function MOYENNE_PERSO(X As Range) As Single
  Dim n As Integer, i As Integer, S As Double
  n = X.Cells.Count
  S = 0
  For i = 1 To n
        S = S + X.Cells(i).Value
  Next
  MoyennePerso = S / n
End Function
```

Cette fonction utilise plusieurs propriétés/méthodes d'une plage de données (Range) :

- `X.Cells.Count` détermine combien il y a de cellules dans la plage de données
- `X.Cell(i)` va désigner la i-ème cellule dans la plage de données. Ici, on suppose (implicitement) que la plage est linéaire, mais si la plage est rectangulaire, alors on peut accéder aux éléments de la plage avec un double indice `X.Cell(i,j)`.
- `X.Cells(i).Value` va alors désigner la valeur de la i-ème cellule de la plage

Globalement, la fonction réalise la somme des cellules de la plage dans la variable `S`, puis divise par le nombre d'éléments de la plage pour avoir la moyenne.

20. Implémenter la fonction ci-dessous dans votre module de fonctions et tester la dans la colonne `E`.

B.2 À vous de jouer

21. Modifier le calcul de la moyenne en création une fonction qui tiendra compte de la pondération :

```
MOYENNE_POND(Notes As Range, Ponderation As Range) As Single
```

22. Modifier la fonction précédente pour tenir compte de l'absence de note. Pour cela, vous pourrez vous inspirer du test ci-dessous qui vérifie si la cellule est vide ou non. Valeurs

```
    If X.Cells(i).Value = vbNullString Then ...
```

Initiation VBA sous Excel

Dans cette initiation au VBA, l'objectif est de découvrir des fonctionnalités de base qui peuvent être faites à l'aide des macro VBA (procédures).

Objectifs (savoir-faire en VBA, par une procédure) :

- Lire des données dans une feuille
- Traitement de ces données
- Placer les résultats dans une feuille
- Mise en forme des cellules d'une feuille

1 *Exemple illustratif des fonctionnalités*

On vous invite à avoir sous les yeux le référentiel des fonctions utiles de VBA ainsi que le cours. Vous y trouverez les informations utiles sur les fonctions à utiliser.

	A	B	C
1	noms	taille	
2	albert	170	
3	raymond	175	
4	georges	162	
5	bruno	183	
6	jean	170	
7	hugo	178	
8	eudes	189	
9	alceste	176	
10			
11			
12			
13			

1. Dans la feuille 1 du classeur actif, entrer les valeurs ci-contre (ou d'autres si vous le souhaitez).

Dans notre exemple, vous allez calculer la moyenne et la variance de la série de valeurs de la colonne « taille ». Pour cela, il nous faut une macro qui « lit » les valeurs dans la feuille 1, calcule la moyenne, la variance et l'écart-type, et écrit les résultats dans la feuille 2.

2. Créer une nouvelle macro nommée `CalculMoyVar`.

A *Définition de variables*

Avec le code ci-contre, vous aller définir des variables correspondant à :

- La feuille 1
- La feuille 2
- Le nombre de valeurs de la série
- La moyenne de la série
- La variance de la série
- L'écart-type de la série

```
(Général)
Sub CalculMoyVar()

'déclaration des variables utilisées par la suite
Dim Fdata As Worksheet
Dim Fresult As Worksheet
Dim SomX As Double
Dim SomX2 As Double
Dim N As Integer
Dim SCE As Double
Dim Moy As Double
Dim Variance As Double
Dim EcartType As Double

End Sub
```

On rappelle qu'une variable a un nom *(X, Y, Taille, Nom)*, un type et un contenu. Il est préférable (mais pas indispensable en VBA) de déclarer une variable avant de l'utiliser. Ceci se fait avec l'instruction `DIM`. Cette instruction réserve un emplacement en mémoire pour la variable déclarée. On peut imaginer une variable comme un « récipient » qui a un nom et un contenu.

Il est évident qu'on ne peut pas mettre dans une variable des valeurs d'un autre type. Par exemple, le code ci-dessous produit une erreur car on essaie de mettre une chaîne de caractères dans une variable de type numérique.

```
Dim X as Integer

X = « Hello »
```

Il faut donc faire attention lorsqu'on veut mettre le contenu d'une cellule dans une variable. L'exemple ci-dessous produira une erreur si la cellule contient du texte

```
Dim X as Double

X = mafeuille.Cells(1,1).value
```

Remarques :

- Le nom d'une variable doit commencer par une lettre, et ne doit ensuite contenir que des lettres ou des chiffres (surtout pas d'espace, ni de lettre accentuées comme é, à, …)
- Les lettres minuscules ou majuscules ne sont pas différenciées
- Il est conseillé de donner des noms qui évoquent le contenu de la variable (ex : Moy), mais ce n'est pas obligatoire. Cela facilite simplement la lecture du programme

Nous allons maintenant donner des valeurs à ces différentes variables.

3. Ajouter les instructions ci-dessous pour attribuer les feuilles du classeur aux variables `Fdata` et `Fresult`

```
Set Fdata = ActiveWorkbook.Sheets(1)
Set Fresult = ActiveWorkbook.Sheets(1)
```

4. Ajouter ensuite les deux instructions ci-dessous qui permettent de savoir combien il y a de lignes au tableau (NB : il est préférable de calculer cela à partir des données que de mettre une constante « en dur » dans le programme au risque de perdre en généricité)

```
NbreLigne  = Fdata.UsedRange.Rows.Count
NbreCol  =  Fdata.UsedRange.Columns.Count
```

Il nous faut maintenant calculer `N` (le nombre de valeurs) : c'est le nombre de lignes – 1 (la première contient du texte correspondant aux noms des variables)

```
N = NbreLigne  - 1
```

B Calcul et affichage de la moyenne

Il nous faut maintenant parcourir la colonne 2 de la feuille `Fdata` où se trouvent les valeurs dont on veut calculer la moyenne, variance et écart-type. On va commencer par faire le code correspondant au calcul de la somme. Pour effectuer ce type d'opération, tous les langages utilisent des « boucles » dans lesquelles s'exécutent des instructions répétées.

Pour le calcul de la somme, nous avons besoin d'une variable annexe Sum qui va accumuler progressivement les valeurs parcourues.

```
Dim Sum As Double
Sum = 0
For i = 2 To NbreLigne
   Sum = Sum + Fdata.Cells(i, 2)
Next i
```

5. Ajouter ce code et compléter le pour calculer la moyenne dans la variable `Moy` (en utilisant la variable `N`).
6. Finalement, mettre l'affichage de la moyenne dans la feuille 2 avec le code ci-dessous

```
Fresult.Cells(1, 1) = "Moyenne:"
Fresult.Cells(1, 2) = Moy
```

C Calcul et affichage de la variance

7. Faire de même pour le calcul et l'affichage de la variance.

- Vous calculerez tout d'abord la somme des différences au carré entre les valeurs du tableau et la moyenne

`Moy`.

- Ensuite, il faudra, calculer la moyenne pour obtenir la variance

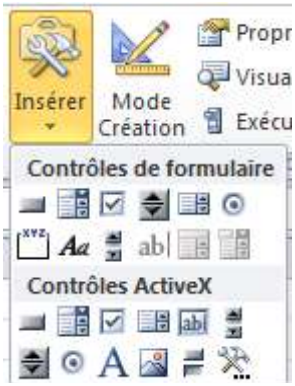
D Interactions avec l'utilisateur

On va commencer par afficher un message pour indiquer à l'utilisateur que le calcul s'est bien terminé. Vous utiliserez pour cela l'instruction suivante à placer en fin de votre procédure :

```
msgbox("calcul terminé")
```

Ensuite, on souhaiterait que cette macro soit exécutée depuis la feuille de calcul plutôt qu'à partir de l'éditeur visual basic.

8. Retourner dans Excel
9. Dans l'onglet **`Développeur`**, utiliser l'icône **`Insérer`** pour ajouter un contrôle de type bouton directement dans votre feuille Excel
10. Il vous propose alors d'affecter une macro à ce bouton. Dans notre cas, comme nous avons déjà créer notre macro, il suffit de la sélectionner dans la liste des macros disponibles, et de valider avec **`Ok`**.
11. Il est alors possible de modifier le texte du bouton lorsque celui-ci est encadré (sinon faire un **`clic-droit>Modifier le texte`**)
12. Tester !



2 Améliorations de la fonction de calcul

A Gestion des valeurs vides

On souhaite maintenant faire en sorte de calculer les moyenne et écart-type pour un tableau pouvant contenir des cellules vides. Il est donc nécessaire de ne pas tenir compte de ces lignes lors de la division par le nombre d'individu. Le calcule de `N` doit donc être revu.

Pour le calcul de la moyenne, on propose donc de procéder comme suit :

```
Sum = 0
N = 0
For i = 2 To NbreLigne
   If Fdata.Cells(i, 2) <> "" Then
      Sum = Sum + Fdata.Cells(i, 2)
      N = N + 1
   End
Next i
```

Le test permet de savoir si la cellule est vide, et si ce n'est pas le cas, alors on incrémente `N` et on ajoute la valeur de la cellule à la somme.

13. Faire de même pour le calcul de l'écart type

B Mise en forme des cellules de résultat

14. Ajouter le code suivant qui modifie les propriétés graphiques des cellules de résultat pour les mettre en gras bleu, sur fond rouge

```
With Fresult.Range("A1:B2")
   .Borders.LineStyle = xlContinuous
   .Interior.Color = RGB(255, 0, 0)
   .Font.Bold = True
   .Font.Color = RGB(0, 0, 255)
End With
```

L'utilisation de `With` pour tout un range de cellule permet d'économiser un grand nombre de ligne de programme ! C'est particulièrement utile lors de la mise en forme automatique de vos tableaux.

Itérations avec VBA

Ce TP est dédié à automatiser des traitements sur tout un tableau, ligne par ligne, ou feuille par feuille.

Objectifs (savoir-faire en VBA) :

- Généralisation d'une expression Excel en VBA
- Calculer les valeurs vs générer des formules de calcul
- Itérer sur un tableau de données
- Itérer sur des feuilles

Attention : pas de `Ctrl+Z` avec les macros !!

Contrairement à la plupart des manipulations sous Excel, les effets de l'application d'une macro ne peuvent être annulés ! Il faut donc faire très attention à vos formules !

Il est donc impératif d'enregistrer très régulièrement votre travail, voire de travailler sur une copie que vous mettrez à jour une fois certain de votre travail.

On commence par rappeler que la structure d'une telle macro sera de la forme :

```
Dim ligne As Integer
For ligne = 1 To 100
        Range("A" & ligne).value = ligne
Next ligne
```

1. Dans une feuille vide, implémenter cette macro puis testez la.

Par la suite, on va s'appuyer sur un exemple de document Excel pour la gestion de notes d'un professeur. Télécharger pour cela le fichier `Notes_ReplFormules_base.xlsm`. La feuille de base est fonctionnelle. L'objectif du TP va être de remplacer les formules actuellement en place par des macros.

1 *Calculer la moyenne à l'aide de VBA*

Sur la feuille `Notes`, chaque étudiant à différentes notes obtenues pendant le cours. La moyenne, colonne `H`, est actuellement réalisée par des opérations arithmétiques. L'objectif de cette section va être de refaire les mêmes calculs dans la colonne `J`.

On va remplacer le calcul de la moyenne de chaque étudiant de la feuille `Notes` par une macro VBA créée de toute pièce. On veut faire le calcul directement dans le programme (et donc cacher le calcul de la moyenne à l'utilisateur). L'intérêt de faire cela réside principalement dans le fait de pouvoir faire des formules complexes en évitant d'avoir à se confronter à la syntaxe parfois délicate des formules Excel.

Dans le principe, toute formule Excel peut se traduire par des calculs en programmation VBA. Par exemple, on peut noter qu'une formule « `=H10 * F4+3` » dans la cellule « `E10` » peut être transformée en une macro[1] :

```
Range("E10")=Range("H10")*Range("F4")+3
```

2. Créer une nouvelle macro `CalculMoyenne()`
3. En observant la formule de la cellule `H5`, et en s'inspirant de l'exemple ci-dessus, reproduire le même calcul dans la macro VBA en mettant le résultat dans la cellule `J5`, et tester votre macro

1 Ou encore `Cells(10,5)=Cells(10, 8)*Cells(4,6)+3`, mais ici, on reste sur l'utilisation des références aux cellules à partir de la fonction `Range`.

Dans le cas de cette formule, vous pouvez très bien avoir un mélange entre l'utilisation de la fonction `Range` pour les cellules qui seront fixes lors de la recopie (traditionnellement avec des `$`) et celles qui seront adaptées par recopie.

L'objectif de la suite du travail est d'itérer ce calcul sur toutes les lignes. Ceci va être possible grâce à l'utilisation d'une itération (boucle `For`) et l'utilisation d'un indice de ligne, c'est à dire une variable que l'on va appeler **`ligne`**, pour parcourir chacune des lignes, une à une comme illustré dans le code de la première page.

Les inscriptions VBA qui ont été écrites pour la cellule `J5` doivent donc être généralisées pour être faite pour une ligne au numéro **`ligne`**. Chaque référence à une cellule de la ligne 5 va ainsi être modifiée de la sorte :

- Le code `Range("J5")` va devenir `Range( "J" & ligne )`

Dans le code modifié, le nom de la cellule utilisée par la fonction range est une chaîne de caractères composée par `"J"` et le numéro de la ligne. Pour ce TP, j'indique en bleu les chaînes de caractères pour faciliter la compréhension de ces syntaxes.

4. Commencer par définir une variable **`ligne`**, puis donner la valeur 5 à cette variable. Modifier ensuite votre formule précédente pour qu'elle généralise le calcul de la moyenne pour une ligne **`ligne`**. L'exécution de cette nouvelle macro devrait fonctionner à l'identique.

Finalement, pour l'utilisation de la boucle `For`, il faut encore savoir quelles sont les bornes (le début et la fin) des lignes sur lesquelles itérer.

5. Modifier maintenant votre macro pour qu'elle itère votre formule à l'aide d'une boucle `for` sur toutes les lignes de votre tableau (vous adapterez la formule ensuite). Pour cela, vous aurez besoin :
   1. du numéro de ligne par lequel commencer (a priori, c'est 5)
   2. du numéro de ligne auquel arrêter les traitements : ce numéro doit être obtenu automatiquement pour s'adapter correctement au nombre réel d'étudiants (utiliser la méthode du cours en vous basant sur la colonne `A`)

2 *Macro de génération de formule*

Une limitation de la macro précédente est d'avoir à réexécuter la macro lorsque le professeur change une note pour mettre à jour les moyennes, alors qu'elles étaient adaptées dynamiquement avant.

Dans cette partie, l'idée est donc de réaliser une macro qui va générer des formules dans les cellules du tableau Excel, plutôt que des valeurs elles-mêmes.

6. Créer une nouvelle macro `CalculMoyenneAuto()`

De même que précédemment, on va d'abord travailler sur une cellule (la `K5`), puis on généralisera progressivement.

7. Dans cette macro, modifier la cellule `K5`, pour y placer la formule de la cellule `H5` (vous pouvez mêmes copier-coller la formule). Ici, on veut que le programme VBA ajoute une formule dans la feuille de calcul (pas la valeur de la moyenne), il faut donc que toute la formule soit entre guillemet (du point de vue de VBA, c'est comme si vous mettiez du texte dans une cellule).
8. Tester votre macro.

Q.1) Quel est le problème rencontré ? (cf cours)

9. Corriger et tester votre macro jusqu'à son fonctionnement pour cette cellule !

Il faut maintenant adapter la formule pour qu'elle prenne en compte le numéro de ligne, lorsque nécessaire. La transformation qui avait été faite pour généraliser une référence à une cellule peut être effectuée de manière similaire pour une formule : les numéros de lignes qui doivent changer dans les calculs sont remplacés par des variables.

10. Dans la formule, remplacer le numéro de ligne par la variable qui a servi à itérer (disons qu'elle s'appelle toujours **`ligne`**). En simplifiant un peu votre formule, le code :

```
Range("K5").Formula = "=(E5*E$4 + F5*F$4) / (E$4 + F$4)"
```

va devenir quelque chose comme :

```
Range("K" & ligne).Formula = "=(E" & ligne & "*E$4 + F" & ligne & "*F$4) / (E$4 + F$4)"
```

> Les espaces autours de `&` semblent importants pour que VBA comprenne cette ligne !

11. En reprenant la même stratégie que la partie précédente (utilisation de la boucle `For`), automatiser le calcul pour toutes lignes du tableau
12. Exécuter votre macro et vérifier que les formules obtenues dans la feuille sont correctes

Quelques remarques :

- On aurait pu utiliser la fonction `AutoFill` de VBA pour simuler la recopie vers le bas de la formule créée dans la cellule `K5`.
- La recopie d'une formule ressemble plus à un exercice de programmation qu'à un vrai exemple d'application… la recopie de la formule dans Excel peut sembler plus efficace, mais on verra dans la seconde partie du TP qu'il peut être utile de créer une feuille de calculs sans que l'utilisateur ait à interagir avec la feuille.
- Quelle stratégie est la meilleure : automatiser les calculs ou automatiser la création des formules ? En fait, cela va dépendre de vos besoins, il faudra s'adapter !

3 *On recommence …sur la feuille TP1*

La feuille TP1 contient des notes qui ont été saisies sous la forme de lettres (A, B, C, D, etc). La formule de la colonne `C` utilise la fonction `RechercheV` pour établir la correspondance entre les lettres et une valeur numérique (à partir d'un tableau annexe).

Nous prenons cette feuille pour pratiquer les mêmes manipulations sur une fonction Excel un peu plus avancée (la fonction `RECHERCHEV`). En particulier, les fonctions utilisées dans les formules Excel existent également directement dans le langage VBA, ce qui permet de les utiliser dans vos calculs.

13. Créer une macro `CalculTP` qui contiendra les opérations pour une feuille de notes

A *Calculer les notes*

Dans cette partie, vous utiliserez la première stratégie pour calculer les moyennes en programmant les calculs en VBA. La formule Excel `=RechercheV(B5;F$15:G$19;2)` peut se traduire en VBA par le calcul suivant :

```
Application.Vlookup( Range("B5"), Range("F$15:G$19"), 2, False)
```

On retrouve donc quasiment la même syntaxe, mais à la place de mettre directement le nom des cellules, il faut mettre des « objets » qui représentent les cellules à traiter.

Le résultat de cette opération sera alors placé dans la cellule `C5`.

Q.2) Comment généraliser cette formule pour qu'elle fonctionne sur une ligne numéro **`ligne`***?*

14. Procéder de la même manière que précédemment pour remplir automatiquement les valeurs de la colonne `C`

B *Générer les formules de calcul de notes*

Procéder maintenant selon la seconde stratégie de calcul pour avoir formules dans la colonne `D` qui soient automatiquement générées. La formule de la cellule `D5` sera donc `=RechercheV(B5;F$15:G$19;2)`.

Q.3) Comment généraliser cette formule pour qu'elle fonctionne sur une ligne numéro **`ligne`***?*

15. Dans votre macro, généraliser la formule puis itérer pour qu'elle soit recopiée sur toute les lignes de la colonne `D`.

Une fois votre macro testée, vous pourrez la tester sur les autres feuilles `TP2` et `Devoir`, dont l'organisation générale est identique à `TP1` (on a simplement adapté la valeur numérique des lettres).

4 *Itération sur les feuilles*

On aimerait maintenant appliquer la même macro sur toutes les feuilles d'un coup … ce serait tout de même plus pratique que d'avoir à le faire sur chacune d'elles !

De la même manière qu'on traitait successivement les lignes d'un tableau, il est possible de traiter les feuilles d'un tableau.

16. Commencer par copier le code ci-dessous dans une macro `CalculTout`, analyser le puis tester le.

```
Dim f As Integer
For f = 1 To Worksheets.Count
        MsgBox("Feuille " & Worksheets(f).Name )
Next f
```

17. Modifier votre macro `CalculTout` pour qu'elle calcule les notes de chaque feuille. En complément de l'exemple ci-dessus, vous aurez besoin d'activer une feuille avec l'instruction du type `Worksheets(f).Activate`. En utilisant l'instruction `CalculTP` vous « lancerez » votre macro réalisé précédemment, sur la fenêtre active !

Attention : la première feuille ne devra pas être traitée pas la macro `CalculTP` !

18. Compléter maintenant votre macro pour qu'une fois toutes les feuilles de notes traitées, vous reveniez à la première feuille pour mettre à jour les moyennes en utilisant votre macro `CalculMoyenne()`

Gestion d'une feuille de notes en VBA

Dans ce TP, on fournit la feuille de notes d'un professeur qui doit calculer les moyennes obtenus au module à partir de plusieurs notes.

Pour faciliter sont travail de gestion de notes, on souhaite donner la possibilité à l'enseignant de réaliser en 1 clic toutes les opérations suivantes :

- ajouter une nouvelle feuille pour un nouveau TP
- pré-remplir la feuille créée avec les noms des étudiants
- insérer les formules adéquates pour reporter les notes de cette feuille dans feuille de notes principale
- adapter la feuille de note principale pour qu'elle calcule correctement les moyennes (pondérées) de chaque étudiants

Un modèle de l'état final de la feuille est donné dans le fichier `Notes_etatfinal.xlsm`.

1 Ajout automatique d'une feuille de calculs

A Ajouter une feuille

1. Ouvrir le fichier contenant les informations de base : `Notes_base.xlsm`.
2. Ajouter un bouton en haut de la feuille qui servira de contrôle pour lancer une macro que vous nommerez par exemple `AjouterTP`

L'objectif est maintenant de compléter le code pour ajouter une nouvelle :

3. Commencer par enregistrer une nouvelle macro en réalisant les opérations suivantes
   1. insérer une nouvelle feuille
   2. renommer cette feuille (double cliquer sur l'onglet du bas et éditer le directement)
   3. ajouter les entêtes du tableau de notes (noms, prénom et notes) à la ligne 3
   4. stopper l'enregistrement
4. Recopier maintenant la partie du code qui vous intéresse dans votre macro puis nettoyez le

On a maintenant une bonne base de travail qu'il va falloir adapter pour que cela fonctionne, cette étape est certainement la plus délicate. Pour y arriver, n'hésitez pas à faire des recherches sur internet !

5. Adapter le code :
   1. Dans un premier temps, il faut faire en sorte que l'ajout de la feuille se fasse plutôt en fin (et pas après la feuille courante, comme cela doit être actuellement le cas). La transformation du code doit ressembler à quelque chose comme :

```
Sheets.Add After:Worksheets(sheetnumber)
```

Dans ce code, sheetnumber correspond au nombre de feuille qu'il y avait avant l'insertion de la nouvelle feuille. Il peut s'obtenir grâce à l'instruction `Worksheets.count`.

   2. Adapter le nom de la feuille pour que celui-ci corresponde au numéro de la feuille qui a été insérer (vous pourrez réutiliser la variable `sheetnumber`)

B Recopie de la liste des étudiants

Vous allez maintenant recopier les noms et prénoms des étudiants dans la nouvelle feuille. Pour simplifier le travail, la liste commencera à la ligne 6, comme dans la feuille principale.

6. Construire une variable `nbetudiants` qui détermine la dernière ligne remplie du

tableau de la feuille principal (NB : dans la mesure où la liste commence à la 6eme ligne, il ne s'agit pas du nombre d'étudiants, comme le nom de la variable le suggère!!)

7. En utilisant une boucle, recopier les colonnes de noms et prénoms de la feuille principale dans votre nouvelle feuille.

C Report des notes dans la feuille principale

Dans la feuille principale, il faut ajouter une nouvelle colonne qui va contenir les notes de la feuille qui vient d'être remplie (notes qui seront remplies plus tard par le professeur dans la colonne `C`). Pour cela, on va ajouter une colonne dans la feuille principale, à droite du tableau actuel, dont la formule sera très simple (voir l'état final attendu si besoin).

De la même manière que précédemment, on vous proposer de commencer par enregistrer manuellement une macro, puis d'adapter le code générer à votre besoin.

8. Enregistrer une macro dans laquelle vous ajouterez la nouvelle colonne, son entête, le coefficient de cette note (1 par défaut) et ajouterez une formule pour recopier les valeurs de la colonne `C` de la feuille TP.
9. Nettoyer le code et copier/coller le dans votre macro `AjouterTP` à la suite des instructions précédentes (dans le temps)
10. Adapter le code … pour cela vous aurez certainement besoin de répondre à la bonne question suivante.

Q.1) Sachant qu'on ajoute la seconde feuille, quel est le numéro de la colonne à ajouter au tableau (en fonction de `sheetnumber`) ?

D Ajout d'une formule pour le calcul de la moyenne dans la feuille principale (*)

La formule (recopiable) pour calculer la moyenne du premier étudiant est de la forme où les cellules `E$4:G$4` contiennent les coefficients de pondération des différentes notes. :

```
=SOMMEPROD(E6:G6;E$4:G$4)/SOMME(E$4:G$4)
```

Dans cet exemple, on a considéré qu'il y avait 3 notes : ceci explique le choix d'avoir une formule dont la plage de données à considérer s'arrête à la colonne `G` … mais si on ajoute une nouvelle note, il faut que la formule prenne en compte la colonne `H`.

On veut ici que les cellules contiennent des formules et pas des valeurs, de sorte que les résultats soient mis à jour au fur et à mesure de l'ajout des notes sans avoir à faire aucune autre manipulation.

Il faut donc créer en VBA une formule qui tienne compte de tout cela ! La difficulté est de gérer correctement les indices. Vous avez deux possibilités : la version simple utilise une itération pour faire une formule spécifique à chaque ligne, la version plus complexe consiste à créer une formule recopiable et la recopier en VBA (cf TP précédents). La version avec recopie est difficile : la formule fait intervenir des références absolue et évolutive, ce qui nécessite d'avoir les idées très claire pour y arriver !

Personnellement, j'ai utilisé la version avec recopie en ayant recours aux nommages de position R1C1 (https://www.formuleexcel.com/references-relatives-et-absolues-en-vba/)

E Mise en forme de la nouvelle colonne de la feuille principales

Toujours en commençant par un enregistrement de macro pour récupérer les noms de fonction spécifique à Excel, puis en adaptant ce code, ajouter le formatage automatique de la nouvelle colonne de notes :

- recopier les propriétés de formatage pour l'entête
- mettre des nombres avec 2 chiffres après la virgule
- Ajouter un quadrillage sur l'ensemble des cellules

Automatiser des graphiques avec VBA

Dans ce TP, l'objectif est d'automatiser la création de graphiques à l'aide de VBA. La méthode choisie consiste ici à utiliser le créateur de macro pour créer une première macro, puis de l'adapter aux besoins spécifiques.

Objectifs (savoir-faire en VBA) :

- Lire des données dans une feuille

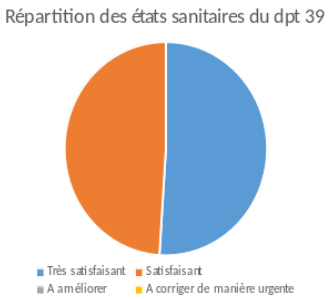
1 *Introduction*

On vous fourni des données de base dans le fichier `DonneesAlim_base.xls`, il s'agit de données réelles d'évaluation sanitaire de restaurants[1]. Chaque feuille correspond aux données d'un département (ici 6 départements ont été extraits).

On s'intéresse à créer des graphiques pour visualiser la répartition dans chaque département des évaluations qui ont été faites sous la forme d'un graphique en « *camembert* » (pie-chart). L'illustration ci-dessous donne le graphique obtenu pour le département 39 pour lequel seules deux des quatre modalités sont réellement présentes dans les données.

On veut obtenir un tel graphique pour chacun des 6 départements (et potentiellement généraliser à tous les départements). La réalisation de ce type de graphique n'est pas immédiate sous Excel puisqu'il s'agit de calculer un histogramme (tableau collectant les nombre de ligne pour chaque modalité)

Vous aller procéder de plusieurs manières dans le but d'arriver au résultat souhaiter !



2 *Réalisation d'un graphique par menu Excel*

Tout d'abord, vous aller pouvoir tester la réalisation d'un histogramme par Excel. Ici, la première limite est qu'on ne va pas pouvoir simplement réalisé un diagramme en camembert avec Excel, mais uniquement un graphique en barre pour représenter la distribution de modalités (histogramme).

Pour créer un histogramme rapidement :

- sélectionner la colonne « `Synthèse` »
- utiliser l'icône **`Insertion>Graphique Recommandé`** pour construire un histogramme groupé
- valider pour finaliser la création de l'histogramme

Excel créé alors une nouvelle feuille sur laquelle il a en fait créé un tableau croisé dynamique associé un à graphique qui représente l'histogramme. Nous avons donc ici l'inconvénient majeur de cette solution qui impose de créer une nouvelle feuille alors que nous souhaiterions que le graphique s'affiche directement sur la feuille des données.

Cette solution ne répond donc pas à notre attente et nécessite donc de faire un peu de travail pour obtenir le résultat souhaité.

3 *Utilisation de formules « classiques » dans Excel*

La solution que nous proposons pour obtenir notre graphique est dans un premier temps de le réaliser à l'aide de formules sous Excel. Pour cela il faut 1) créer un tableau qui va réaliser le décompte de chaque modalité (cf illustration) puis 2) créer ensuite un graphique à partir de ce nouveau tableau.

1. Vous allez donc commencer par créer le tableau collectant les comptes des modalités :

Très satisfais	25
Satisfaisant	24
A améliorer	0
A corriger d	0

1 Ces données sont disponibles sur le site https://www.data.gouv.fr

1. Utiliser la feuille « `35` » pour créer votre premier graphique
2. Dans les cellules `P7` à `P11`, recopier les noms des quatre modalités. On les trouve dans la feuille « `35` » elles peuvent donc être copiées/collées pour être sûr d'avoir exactement les bons textes.
3. Dans la cellule `P8`, utiliser la fonction `NB.SI` pour compter les nombres de cellules de la colonne `G` qui ont la valeur de la cellule `P7`
4. Recopier votre formule vers le bas.

2. Vous pouvez maintenant créer le graphiques
   1. Sélectionner le tableau de données
   2. Dans le ruban **`Insertion`**, utiliser l'icône graphique pour la création d'un camembert
   3. Ajuster le titre de votre graphique (vous noterez que ce titre comporte le nom du département)

Vous pourriez donc procéder de la même manière pour toutes les feuilles, mais ce serait un peu fastidieux à faire pour tous les départements. Ce qui serait bien ce serait de pouvoir automatiser la répétition de ces opérations … par exemple avec une macro.

4 Réalisation d'une macro

A La macro brute

Maintenant que vous savez assez précisément ce que vous voulez faire et comment le faire, vous allez pouvoir créer une macro pour enregistrer cette « procédure » de création d'un graphique :

3. Commencer par supprimer votre réalisation précédente (il va falloir la refaire intégralement et avec le moins de fioriture possible)
4. Lancer l'enregistrement de votre macro et donner le nom `CreerGraphique()` à votre macro
5. Réaliser de nouveau les opérations de la section précédente
6. Arrêter l'enregistrement de la macro !

Et voilà, votre macro est prête à être exécuter de nouveau sur une autre feuille !

B Consulter et tester de votre macro

Nous allons commencer par consulter le code qui a été produit par l'enregistrement en ouvrant la console VBA (combinaison de touches **`Alt+F11`**).

> À ce stade, il est possible que vous ayez des choses très différentes les uns des autres ! Je suppose que vous avez néanmoins suivi le détail de la section précédente.

7. Allez ensuite sur une nouvelle feuille (sans graphique) puis lancer votre macro (en plaçant votre curseur dans votre macro puis en utilisant la touche **`F5`** ou bien l'icône à la flèche verte)
8. Constater les dégâts !!

Q.1) Que se passe-t-il ?

En gros, Excel réalise de l'apprentissage « par coeur » de ce qu'il faut faire, et si on regarde attentivement le code, vous devriez y trouver des comportements qu'il va reproduire mais ne correspondent pas vraiment à votre attente. Plusieurs erreurs devraient se produire (essayer de les comprendre avant de passer à la suite) :

- le tableau à bien 4 lignes, mais pas les bonnes modalités (il y a des répétitions)
- il y a un graphique, mais c'est le même que pour le département 35, est-ce un hasard ?
- la légende évoque toujours le département 35
- il annonce des erreurs

C Amélioration du code

Plusieurs améliorations sont à apporter :

9. Nettoyer le code de toutes les actions inutiles … il y a certainement beaucoup de `Select` qui ne servent à rien … il faut tâcher de bien comprendre ce qui s'est passé !
10. Il faut ensuite refaire proprement le code pour la création du tableau de décomptes
    1. Tout d'abord, modifier la création du tableau de décompte (`P7:Q11`) en créant la première colonne par des instructions du type :

```
Range("P7").Value = "Très satisfaisant"
```

    2. Vous pouvez reprendre la formule pour faire les décomptes (inutile de faire un programme pour cela)

```
Range("Q7").Formula = "=COUNTIF(C[-10],RC[-1])"
```

    3. Il y a plusieurs solution pour les cellules `Q8:Q10` qui contiennent la même formules … à vous de voir !
11. Reprendre la partie de création d'un graphique
    1. faites attention à ce que le graphique utilise bien les données de la feuille courante `ActiveSheet.Range("$P$7:$Q$10")` et pas toujours celles de la feuille nommée « `35` ».
    2. Adapter le nom du graphique en combinant celui-ci avec le nom du département (utiliser l'opérateur & pour concaténer des chaînes de caractères). Le département peut être récupéré, par exemple de la manière suivante :

```
Dim dpt As String
dpt = Left(Range("E2").Value, 2)
```

Pour cet exercice, vous ne vous soucierez pas de la position des graphiques. La manipulation de ces graphiques nécessite d'aller chercher des fonctionnalités qui sont un peu spécifiques. La section 6 du TP vous invite à une solution qui fait appel à ces fonctionnalités.

> N'hésitez pas à tester régulièrement vos modifications pour vérifier si elles se comportent comme attendu … vous pouvez procéder par essai-erreurs.

12. Une fois votre procédure mise en place, testez là successivement sur chaque feuille pour y créer le graphique systématiquement.

5 Automatiser le traitement de toutes les feuilles

Votre procédure permet maintenant de traiter une feuille … plutôt que de passer manuellement sur chacune des feuilles, automatisons cette opération !!

Votre procédure fonctionne sur la feuille courante (celle qui est dite `Activate`) pour traiter chacune des feuilles, il faut donc que le programme parcourt chacune des feuilles, l'active et exécute votre procédure.

13. Réaliser une procédure `CreerGraphiques()` qui traite l'ensemble des feuilles (inspirez vous du TP précédent pour le parcours des feuilles de calculs d'un classeur)

6 Tous les graphiques regroupés

Dans cette partie, l'objectif est de faire en sorte d'avoir tous les graphiques regroupés sur une seule feuille. Pour la suite de cette section, on part du code ci-dessous qui réalise la création d'un graphique à partir d'une feuille active sur cette feuille active.

```
Sub CreerGraphique()
    Range("P7").Value = "Très satisfaisant"
    Range("P8").Value = "Satisfaisant"
    Range("P9").Value = "A améliorer"
    Range("P10").Value = "A corriger de manière urgente"
```

```
    Range("Q7").FormulaR1C1 = "=COUNTIF(C[-10],RC[-1])"
    Selection.AutoFill Destination:=Range("Q7:Q10"), Type:=xlFillDefault

    ActiveSheet.Shapes.AddChart2(251, xlPie).Select
    ActiveChart.SetSourceData Source:=ActiveSheet.Range("$P$7:$Q$10")

    Dim dpt As Integer
    dpt = ActiveSheet.Name
    ActiveChart.ChartTitle.Text = "Répartition des états sanitaires du dpt " & dpt
End Sub
```

Dans le cas de la création de tous les graphiques sur une unique feuille, nous allons être très dérangé par l'utilisation d'une feuille active. Ce qu'on aimerait arriver à faire faire à Excel c'est : sur une feuille vierge (la feuille active), générer un graphique sur cette feuille à partir des données de la feuille X.

A Générer le graphique de la feuille « 35 » à partir d'une autre feuille

Nous allons donc commencer par faire l'opération pour la feuille `36` avant de généraliser l'opération sur n'importe quelle feuille.

14. Créer une nouvelle feuille vierge *en dernière position* dans votre classeur et nommée la par exemple « `Graphiques` »
15. Dans le code VBA, copier vous fonction `CreerGraphique()` (ou celle-ci-dessus) dans une nouvelle fonction nommée par exemple `Graphique()`.

On veut faire en sorte que cette dernière macro soit exécutée depuis la nouvelle feuille vierge, créer le tableau de données dans la feuille « `36` » et ajoute le graphique dans la feuille `Graphiques`.

Actuellement, toutes les fonctions de type `Range("X3")` désigne implicitement la cellule `X3` de la feuille active. Il faut donc être plus précis en indiquant qu'on souhaite que ce soit la cellule `X3` de la feuille `36`. Deux solutions s'offrent à nous : utiliser le nommage des cellules sous la forme `'36'!X3`, ou préciser que la fonction `Range` est utilisée sur un objet feuille spécifique. Nous adoptons ici la seconde solution.

16. En début de la fonction `Graphique()`, ajouter le code ci-dessous pour créer une variable qui désigne la seconde feuille de calcul (si on veut la troisième, on mettra 3 à la place du 2, etc)

```
Dim S As Worksheet
Set S = ActiveWorkbook.Sheet(2)
```

Si on veut modifier le contenu de la cellule `P7` de la seconde feuille, on peut alors utiliser l'instruction suivante :

```
S.Range("P7").Value = "Très satisfaisant"
```

17. En analysant votre code pour savoir sur quelle feuille est réalisée chaque action (feuille du département ou feuille `Graphiques`), modifier la fonction `Graphique()` pour qu'elle créer le tableau sur la feuille du département de la 2eme feuille et le graphique sur la feuille `Graphiques`. Cette dernière sera supposée être la feuille active
18. Tester votre fonction graphique à partir de la feuille `Graphiques`. Si besoin, corriger jusqu'à son bon fonctionnement.

B Généralisation pour la création de tous les graphiques

La fonction Graphique() permet actuellement de réaliser une opération pour une feuille donnée (la feuille 2 dans l'exemple). On aimerait que cette fonction puisse fonctionner sur n'importe quelle feuille, pour cela, il est possible d'utiliser la notion de paramètre d'une fonction.

19. Modifier le début de la fonction `Graphique()` pour ajouter en paramètre le numéro de la feuille à traiter :

```
Sub Graphique( fId As Integer )
    Dim S As Worksheet
    Set S = ActiveWorkbook.Sheets(fId)

    (...)
```

Dans le code ci-dessus, nous avons transformé la macro pour lui ajouter le paramètre `fId` qui est une variable locale de la macro dont la valeur sera donnée lors de l'appel de fonction (voir ci-dessous). Cette variable sert dans la fonction à désigner la feuille qui est utilisée par la suite.

En l'état, la macro ne peut plus être exécutée !! En effet, pour quelle fonctionne, il faut lui donner une valeur pour `fId`. Nous avons donc besoin d'une seconde macro.

20. Créer la macro `CreerGraphiquesSurFeuille()` ci-dessous et exécuter cette dernière depuis la feuille `Graphiques`.

```
Sub CreerGraphiquesSurFeuille()
    Call Graphique(2)
End Sub
```

21. En adaptant le code de parcours des feuilles d'un classeur, modifier la macro `CreerGraphiquesSurFeuille()` pour qu'elle créé un graphique pour chaque feuille (sauf la dernière qui sera la feuille `Graphiques`)

C Positionnement des graphiques

Finalement, il est intéressant d'avoir une meilleure gestion des graphiques, on vous propose donc deux bouts de codes.

- Un code pour supprimer tous les graphiques de la feuille active :

```
Dim chtObj As ChartObject
For Each chtObj In ActiveSheet.ChartObjects
    chtObj.Delete
Next
```

- Un code pour repositionner tous les graphiques les uns en dessous des autres :

```
Dim chtObj As ChartObject
x=0
chartHeight = 200
For Each chObjt In ActiveSheet.ChartObjects()
    chObjt.Chart.Parent.Height = chartHeight
    chObjt.Chart.Parent.Width = 300
    chObjt.Chart.Parent.Top = x
    chObjt.Chart.Parent.Left = 100

    x = x+ chartHeight + 10
Next chObjt
```

22. Modifier votre macro pour qu'elle supprime les anciens graphiques, créer les nouveaux, puis les positionnent correctement sur la feuille `Graphiques`.

Création d'un formulaire personnalisé pour Excel

L'objectif de ce TP est de faire un formulaire permettant à une entreprise de consulter et enregistrer les réservations de véhicule de l'entreprise. Le fichier `MacroFormulaireExcel.xls` contient les données de base utiles pour ce TP :

- une liste de véhicule
- une liste de personnel
- des exemples d'enregistrement de réservation

Chacune de ces listes correspond à une feuille de données et les données ont été structurées comme une plage de données permettant les traitements usuels d'*Excel* (et mis en forme automatiquement).

Ce TP explore tout d'abord les fonctionnalité de formulaires automatique de *Excel* puis propose de mettre en place votre propre formulaire personnalisé.

Les compétences à acquérir dans ce TP sont :

- L'utilisation des formulaires
- La configuration de l'environnement *Excel* pour VBA
- La création d'une feuille de formulaire
- La création de macros

1 Configuration d'Excel pour la création de formulaires et de macros

A Personnalisation du ruban

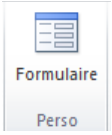
Tout d'abord, nous allons configurer Excel pour la suite du TP. Par défaut, l'interface d'*Excel* ne donne pas accès aux fonctionnalités avancées que nous allons voir dans ce TP. Nous allons donc configurer le ruban (barre des icônes) pour qu'ils répondent à nos besoins.

1. Faire un clic-droit n'importe où sur le ruban, puis utiliser le menu **`Personnaliser le ruban`**

Il y a alors deux listes de fonctions : celle de gauche correspond à l'ensemble des fonctionnalités possibles dans Excel (structurée en catégories), celle de droite correspond à l'organisation du ruban sous forme hiérarchique (ruban, groupe et icônes).

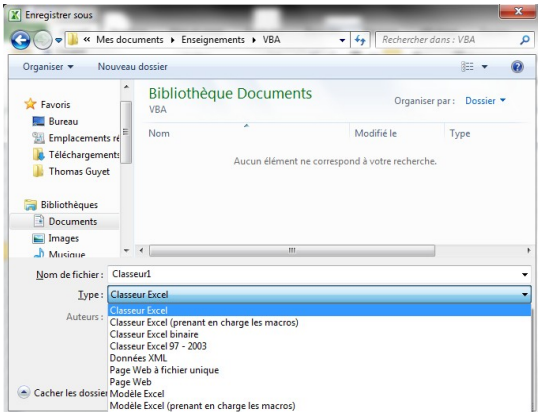
2. Nous allons commencer par ajouter l'icône pour l'accès aux formulaires
   1. Dans la liste de gauche, afficher `Toutes les commandes` (choix dans le sélecteur au dessus de la liste) puis trouver la commande `Formulaires...`
   2. Dans la liste de droite, sélectionner le ruban Données puis créer un nouveau groupe en utilisant le bouton **`Nouveau groupe`** (renommé le ensuite « `Perso` » par exemple)
   3. En utilisant le bouton **`Ajouter`** entre les listes, vous pourrez ajouter la commande de formulaire à le nouveau groupe (il faut à la fois que la bonne commande et le bon groupe aient été sélectionnés!)
3. Valider avec **`ok`**

Si vous consulter à présent le ruban **`Données`**, vous devriez voir apparaître un nouveau groupe contenant l'icône **`Formulaires`**.



B *Enregistrement d'un fichier Excel avec macros*

Dans ce TP nous allons enrichir le fichier *Excel* par des macros (fonctions VBA). Pour que ces macros soient enregistrées avec le fichier, il faut choisir le format de fichier `.xlsm` ! Le format de fichier `.xls` ou `.xlsx` ne conservera pas les macros !



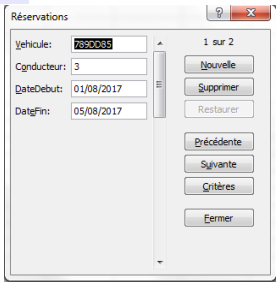
4. Commencer par enregistrer votre fichier avec ce format :
   1. Dans le menu **`Fichier>Enregistrer sous...`**
   2. Choisir le type de fichier « `Classeur Excel (prenant en charge les macros)` » (*cf.* illustration ci-contre)

Ces configurations élémentaires ayant été faites, nous pouvons nous lancer dans la réalisation de notre formulaire.

2 *Création d'un formulaire par défaut*

Dans cette partie, nous testons l'utilisation des formulaires automatiques d'Excel (oui ! Ça existe!!).

5. Création d'un formulaire automatique
   1. Aller sur la feuille réservation
   2. Placer le curseur dans la plage de données
   3. Utiliser l'icône Formulaire nouvellement ajouté dans le ruban **`Données`**



La fenêtre ci-contre s'affiche. Elle permet alors de parcourir et ajouter de nouveaux enregistrements.

6. Tester le formulaire

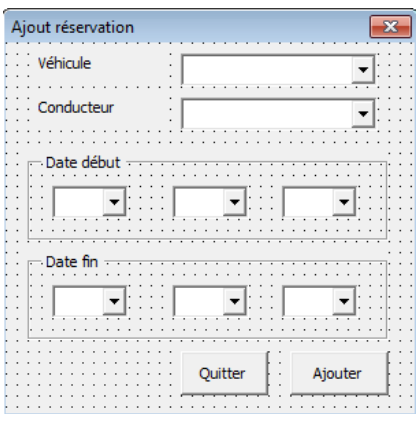
Cet outil a néanmoins pas mal de limite. Ce formulaire serait très bien pour saisir de nouveau véhicules dans la première feuille de ce tableur, mais pour les réservations, il a de nombreuses limites : il ne permet pas de savoir qui est conducteur 3, il ne permet pas de faire une saisie « naturelle » d'une date et n'évite aucune erreur de saisie (par exemple, sur une voiture qui n'existerait pas, saisir autre chose qu'une date, etc.)

Dans la suite, on cherche donc à construire un formulaire qui permettra de réaliser des ajouts de nouvelles réservations de manière plus pratique et plus sûre du point de vue des données. Bien évidemment, il serait préférable d'utiliser *Access* pour créer ces formulaires. Ici, il s'agit plus d'un prétexte à découvrir l'utilisation de VBA qui pourront être utilisés dans d'autres contextes.

3 *Création d'un formulaire personnalisé (une première version)*

A *Création de la feuille d'interface*

7. Dans le ruban **`Développeur`**, utiliser l'icône Visual Basic pour faire afficher l'environnement de programmation VBA
8. Créer une nouvelle interface à l'aide du menu **`Insertion>UserForm`**
9. À l'aide des outils de création de contrôles et de leurs propriétés, créer l'interface ci-contre



Ce dont vous aurez besoin pour créer cette interfaces :

- utilisation de la boîte à outils pour ajouter des contrôles (listes déroulantes également appelées *ComboBox*, des étiquettes et des boutons). J'ai également utilisé un cadre pour les dates
- utilisation de la propriété `Caption` pour modifier les textes qui s'affichent sur ces différents contrôles
- renommage des contrôles de sorte à les identifier facilement (`cmbVehicule`, `cmbConducteur`, `cmbDDJour`, `cmdDDMois`, etc.)
- Pour le contrôle des véhicules vous modifierez la propriété de `Style` en `2 - fmStyleDropDownList` pour interdire à l'utilisateur de saisir manuellement une valeur

Ne passez pas trop de temps à mettre en forme votre interface … ce n'est pas l'objectif de ce TP ! Ces étapes de conception des interfaces peuvent prendre beaucoup de temps lorsqu'on veut faire les choses bien (configurer les transitions entre contrôles, ajouter des *infobulles*, ajouter des contraintes sur les valeurs acceptables, etc.).

10. Tester votre interface avec la touche **`F5`** (ou l'icône de lecture de la barre)

Pour le moment, votre interface ne fait rien, il va donc falloir associer des codes VBA aux différents événements de la vie de cette interface :

- lors du chargement de cette interface, il va falloir « remplir » les *comboboxes* à partir des données qui sont disponibles (notamment les conducteurs et les véhicules)
- lors de l'utilisation du bouton `Quitter`, on souhaite que l'interface se ferme (sans rien de plus)
- lors de l'utilisation du bouton `Ajouter`, on souhaite d'une nouvelle ligne soit ajoutée en fin du tableau de réservation

La suite de cette section sera consacrée à ajouter des codes correspondant aux différents comportements souhaités.

> Vous penserez à tester régulièrement votre interface à l'aide de la touche **`F5`**.

B Bouton Quitter : fermeture de l'interface

11. Dans l'éditeur d'interface, double-cliquer sur le bouton **`Quitter`** pour atteindre le code qui sera exécuté lorsque ce bouton sera cliqué
12. Ajouter le code ci-dessous

```
Private Sub CommandButtonQuitter_Click()
    Unload Me
End Sub
```

Quelques commentaires sur ce code :

- le nom de la procédure `CommandButtonQuitter_Click()` correspond à la procédure qui sera appelée lors d'un événement de `click()` sur le bouton nommé `CommandButtonQuitter`. Vous n'aurez pas nécessairement le même nom de procédure si votre bouton a un autre nom.
- `Unload me` est l'instruction qui indique de détruire (`unload`) un objet. Ici, l'objet à détruire est `Me`, qui désigne l'interface en cours.

C Chargement de l'interface

13. Créer une nouvelle procédure pour l'évènement `Initialize` du formulaire `UserForm1`

Toujours depuis l'éditeur de macro, choisir le contrôle « `UserForm` » dans la liste déroulante de gauche

(une nouvelle procédure associée à l'événement de clic sera créée, mais peu importe ...), puis sélectionner l'événement « `initialize` » : la procédure est alors créée.

14. Créer un code qui ajoute les valeurs 1 à 31 au contrôle des jours, vous aurez besoin pour cela des éléments suivants :

- votre contrôle sera désigné par une expression du type `Me.cmbDDJour` si son nom était `cmbDDJour`
- l'ajout d'un item à un *combobox* se fait grâce à la fonction `addItem` (l'instruction `Me.cmbDDJour.addItem( 45 )` ajoute donc la valeur 45 au contrôle nommé `cmbDDJour`)

15. Faire de même pour les mois (de 1 à 12) et les années (de 2016 à 2019 par exemple)

Bien, maintenant, on va ajouter la liste des véhicules.

16. Ajouter le code ci-dessous

```
Set Ws = Sheets("Vehicules")
For J = 2 To Ws.Range("A" & Rows.Count).End(xlUp).Row
   Me.cmbVehicule.AddItem( Ws.Range("A" & J) )
Next J
```

- `Ws.Range("A" & Rows.Count).End(xlUp).Row` permet de désigner la dernier ligne (non-vide) de la colonne `A`.

17. Adapter le code ci-dessus pour inclure également l'ajout des numéros de conducteurs dans la liste déroulante des conducteurs
18. Tester votre interface

D *Création d'un nouvel enregistrement*

19. Créer une nouvelle procédure pour l'événement de clic du bouton `Ajouter`
20. Reprendre le code ci-dessous qui permet d'ajout des données pour les deux premières colonnes.

```
Set Ws = Sheets("Réservations")
L = Ws.Range("A" & Rows.Count).End(xlUp).Row + 1 'première ligne vide
Ws.Range("A" & L).Value = Me.cmbVehicule
Ws.Range("B" & L).Value = Cint(Me.cmdConducteur)
```

- La seconde ligne permet d'identifier la première ligne pour laquelle il n'y a pas de données : c'est donc sur cette ligne qu'il faut remplir des informations
- La troisième ligne recopie la valeur de la liste déroulante correspondant au véhicule vers la cellule de la colonne `A` sur la nouvelle ligne.
- La quatrième ligne effectue une recopie similaire mais en utilisant la fonction `CInt` qui va convertir le texte contenu dans la liste déroulante en nombre (*Integer*).

21. En adaptant le code précédent, proposer la suite du code qui va ajouter les dates de la réservation. Il faudra combiner les valeurs des différentes listes déroulantes pour « créer » la date puis la mettre en format de date (similairement à `CInt`)

4 Amélioration du formulaire : pour des données plus sûres

A Vérifier que toutes les données ont bien été remplies

Le code ci-dessous permet de vérifier si la liste déroulante du véhicule contient bel et bien une valeur et si ce n'est pas le cas, il affiche un message d'information à l'utilisateur et met fin à la procédure en cours (`Return`).

```
If IsNull(Me.cmbVehicule) Then
    MsgBox ("Vous devez saisir un numéro de véhicule")
    Return
End If
```

22. Ajouter ce code dans le code correspondant à l'ajout d'une enregistrement
23. Compléter ensuite votre code de sorte à ce que la vérification soit faite sur toutes les informations à fournir par l'utilisateur.

On pourrait également ajouter un test portant sur la validité des dates :

- est-ce que la date de retour est postérieure à la date de début ?
- est-ce que la date de réservation est bien postérieure à la date du jour ?
- est-ce que la date de réservation de cette voiture est compatible avec les autres réservations existantes ?

B Ne permettre que des conducteurs qui ont effectivement le permis

Lors de l'ajout des conducteurs, vous avez ajouté la liste entière des conducteurs, hors la table de données de la feuille `Personnel` précise que certaines personnes n'ont pas le permis de conduire. On souhaite dont filtrer ces personnes de sorte à faciliter la saisie et éviter aussi les erreurs.

24. Modifier le code de la fonction d'initialisation du formulaire pour que seuls les personnes qui ont le permis soient effectivement ajoutées à la liste déroulante (utiliser une structure conditionnelle, `if`)

C Nom des conducteurs

On souhaite maintenant avoir une étiquette qui affiche le nom du conducteur qui est sélectionné pour compléter l'information de son numéro (cela rassure lors de la saisie de l'information)

25. Ajouter un contrôle sous la forme d'une étiquette dans l'interface, supprimer son texte et renommer votre étiquette `labelConducteur`

Q.1) Lors de quel événement cette étiquette devra-t-elle changer de valeur ?

26. Créer la procédure associée à l'événement souhaité puis compléter le code

5 Lancement du formulaire depuis la feuille de calculs

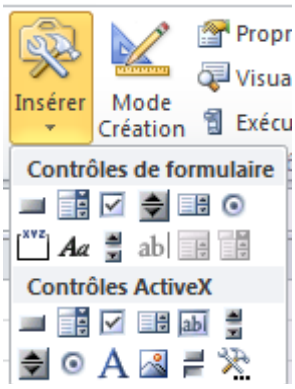
Finalement, nous souhaiterions pouvoir lancer facilement notre formulaire (sans avoir à passer par l'interface développeur d'*Excel*).

27. Créer un nouveau module de macro : depuis l'interface de développement VBA, faire un clic-droit dans la zone de projet VBA puis **`Insertion>Module`** (ou depuis la barre de menu également !). Vous pourrez ensuite renommer votre module.
28. Créer une nouvelle macro en tapant le code ci-dessous

```
Sub OuvrirFormulaire()
   UserForm1.Show
End Sub
```

Cette macro nommée `OuvrirFormulaire` fait juste en sorte d'afficher le formulaire que l'on vient de créer à l'aide de la fonction `Show`.

Cette macro va maintenant pouvoir être utilisée en association avec un bouton que nous allons créer dans la feuille *Excel* des réservations.



29. Fermer l'éditeur VBA et aller dans la feuille `Réservations`.
30. Dans l'onglet **`Développeur`**, utiliser l'icône **`Insérer`** pour ajouter un contrôle de type bouton directement dans votre feuille *Excel*
31. Il vous propose alors d'affecter une macro à ce bouton. Dans notre cas, comme nous avons déjà créer notre macro, il suffit de la sélectionner dans la liste des macros disponibles, et de valider avec **`Ok`**.
32. Il est alors possible de modifier le texte du bouton lorsque celui-ci est encadré (sinon faire un **`clic-droit>Modifier le texte`**)
33. Tester !

6 *Compléments*

Dans le cas de liste déroulante dont le contenu est fixe (par exemple « Mr, Mme, Mlle ») il est possible de créer une tableau avec ses trois valeurs puis de nommer cette zone de données. Ensuite, dans un contrôle de liste déroulante, il suffit de donner le nom de la zone de données dans la propriété `rowsource`.