Mémento VBA SL8 infodauphine.com : Recherches, Tris (au verso) et récursivité

Recherche

Version conseillée : **variable** avec **valeur de départ** *VD* qui change **dans une condition**, et **à la fin** voir si elle a **changé** ou non de valeur (*drapeau*)

Vérifier qu'**au moins une** cellule est positive	Vérifier que **toutes** les cellules sont positives

`Trouvé = False`
Boucle sur les cellules
  Si la cellule **est** positive
  `Trouvé = True`
  Sortie de la boucle
	`ok = True`
	Boucle sur les cellules
	  Si la cellule **n'est pas** positive
	  `ok = False`
	  Sortie de la boucle

Après la boucle : **retourner** `trouvé` / `ok` *ou* `If`/`(Else)` pour **tester** si la variable a **gardé sa valeur de départ** ou non pour faire des **actions** dans chaque cas	

Surtout, **pas de `Else` dans la boucle** quand on recherche/vérifie ! (ERREUR FATALE ET TRES COUTEUSE, toujours se méfiez des `Else` !)

Attention aussi (très important !), pour vérifier que **toutes** les valeurs correspond à un critère (« *elles sont toutes positives* »), on **recherche** en fait une **valeur incorrecte** (c'est seulement si une cellule n'est **pas positive** que je sais qu'elles ne le sont **pas toutes**).

Toujours observer pour **quelle valeur** on a la **réponse** (voir débriefs EasyVBA avec les flèches verte/rouge pour montrer les parcours). Pour **nommer** les variables *drapeau* : utiliser la fin d'une phrase ! Ex : « la valeur est *trouvé* » ➔ `trouvé`, « la vente n'est pas *possible* » ➔ `possible` , « il y a *aucun* positif » ➔ `aucun`

Pour vérifier qu'une valeur existe, mais pouvoir en plus **faire une action sur celle-ci**, se servir d'une **valeur *impossible*** comme **valeur de départ** (par exemple partir de -1 pour la variable qui va stocker le numéro de la ligne de la cellule positive), pour ensuite lui donner la **valeur correcte dans la condition**. Si notre variable reste -1, c'est comme si elle restait `False` : on sait que la valeur n'existe pas. **Eviter** de mettre les actions **dans la boucle** quand on veut les faire **1 fois**.

Faire des **actions** sur **la 1ère** ou **la seule** cellule positive, et une autre action **si elle n'existe pas** (exemple dans une colonne)	Faire des **actions** sur **chaque** cellule positive, et une autre action **si elle n'existe pas**

`nlv= -1`
Boucle sur les cellules avec `nl` le n° de ligne de la valeur du tour
  Si la cellule **est** positive
  `nlv = nl`
  Sortie de la boucle
après la boucle, *soit* ***retourner*** *la valeur de `nlv` (la ligne ou -1), soit :*
Si `nlv` est **resté -1** (on sait que la cellule positive n'existe pas)
  Actions à faire si **aucunes** positives n'existent
Sinon
  Actions à faire **avec la ligne `nlv`** (celle de la positive)
	`Trouvé = False`
	Boucle sur les cellules
	  Si la cellule **est** positive
	  `Trouvé = True`
	  actions à faire **pour CHAQUE** positive
	après la boucle :
	Si `trouvé` est **resté `False`** (`If Not Trouvé` ou `If Trouvé = False`)
	  Actions à faire si **aucunes** positives n'existent

- « pour **la** ligne **qui** a le nom *France* » : Dans le branchement de la **boucle**, **récupérer** juste le numéro de ligne (**`nlp`**) et **sortir**.
  Après la boucle, faire **les actions** sur **cette ligne** `nlp` (afficher les valeurs de cette ligne, la colorier/formater, la parcourir)
  Pour vérifier que France existe, tester **après la boucle** la variable `nlp`, si elle est restée sa valeur de départ -1
- « pour **chaque** ligne où le pays est France » : **Dans** le branchement de la **boucle**, faire **les actions pour chaque** ligne « France ». Pour vérifier qu'on l'a fait pour au moins 1, tester **après la boucle** une variable (`trouvé`) restée `False` ou devenue `True`.

Choix de la boucle pour recherche et vérification dans les lignes

Je sais que les lignes vont de **2 à 10** :	Je sais juste que ça commence ligne 2 (on ne connait **pas la fin**):
`For` `nl = 2 To 10` Fin de boucle : **`Next`** `nl`	**`nl = 2`** puis **`Do While`** `Cells(nl,1).Value <> " "` Fin de boucle : **`nl = nl + 1`** et **`Loop`**

Recherche sur plage ordonnée (croissant, y compris alphabétique, ou décroissant) : on doit **toujours optimiser** les recherches **ordonnées** !

Si je ne connais pas le nombre de lignes :	Sinon (**je connais la fin**) : **recherche dichotomique** (*performance : $Log_2(n)$*), avec variables `début` et `fin` :

recherche classique (je parcours et change une variable quand je trouve) mais **je sors EN PLUS si je dépasse** l'endroit où la valeur peut être (exemple nombres croissants : je **sors** quand je tombe sur 12 alors que je cherche 9, *cf SL8*)	On part d'un `Trouvé=False` ou d'un `nlv=-1`
	Tant qu'il reste **au moins 1 ligne** (`début` ne doit pas être à `fin`), on calcule la ligne `milieu`
	  Si x est la valeur du **milieu**, on sort en changeant la variable (`Trouvé=True` ou `nlv=milieu`)
	  Sinon si x est **plus petit** que la valeur du milieu, elle est dans la **moitié haute** : **`fin`** devient `milieu-1`
	  Sinon x est dans la **moitié basse** : **`début`** devient `milieu+1`

Tirage au sort entre *a* et *b*

: prendre **l'entier** du tirage (intervalle : [0,1[ ) **multiplié** par (**l'étendue + 1**), et **ajouter** la borne **minimale**.

Formule à connaître par cœur : `Int( Rand() * (b-a+1) ) + a` ➔ ex : entre 5 et 15 (donc étendue 10) → `Int( Rand() * 11 )+5`

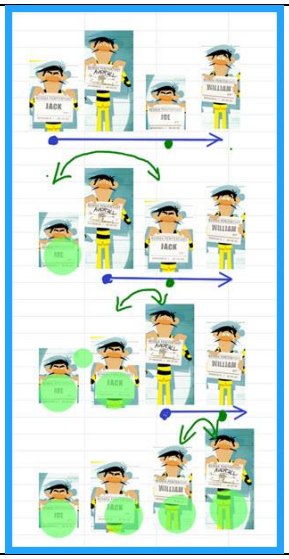
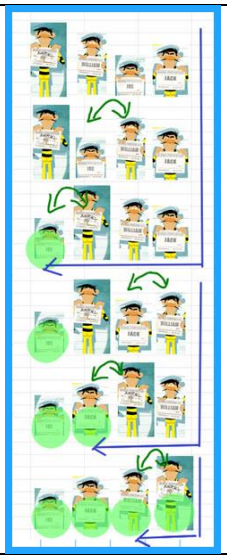
Récursivité

: faire 1 ou 2 **appels** de la fonction/procédure **à l'intérieur d'elle-même**, pour déléguer à **l'appel récursif** une partie du travail

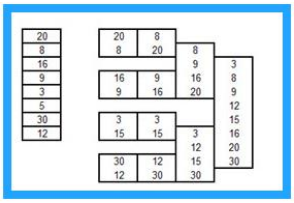
Chaque appel **exécute** au moins **un autre appel** avec au moins **un paramètre différent**. **Logique d'Aller/Retour** *(cf : les rêves dans Inception)* : à **l'Aller**, exécution de tout ce qui est **avant l'appel** et de **chaque appel** jusqu'au **cas d'arrêt**, puis **Retour** avec exécution dans le **sens inverse** de ce qui est **après l'appel récursif**.
Toujours avoir **un cas d'arrêt** où **l'appel récursif** ne sera **pas exécuté** (avec `Exit Function/Sub` , ou mettre l'appel récursif dans `Else` ou dans un `If`)

Exemple (ici fonctions, mais parfois Sub)	**`facto(x)`** : factoriel de x	**`Suite(n)`** : $u_n = 3.u_{n-1}+2.u_{n-2}$ Avec $u_0 = 5$ et $u_1 = 7$	**`trouve(x,debut,fin)`** : recherche dicho de x retourne `True` / `False`
Cas d'arrêt	Si **`x<=1`** (on retourne 1)	Si **`n <= 1`** (on retourne 5 ou 7)	Si **`debut>fin`** (`False`) *ou Si* x **est valeur** du milieu (`True`)
Appel(s) récursif(s)	`facto= x *` **`facto(x-1)`**	`suite = 3 *` **`suite(n-1)`** `_` `+ 2 *` **`suite(n-2)`**	`trouve =` **`trouve( x , milieu+1 , fin )`** si x dans 2ème moitié, sinon : `trouve =` **`trouve( x , début , milieu-1 )`**

Les tris

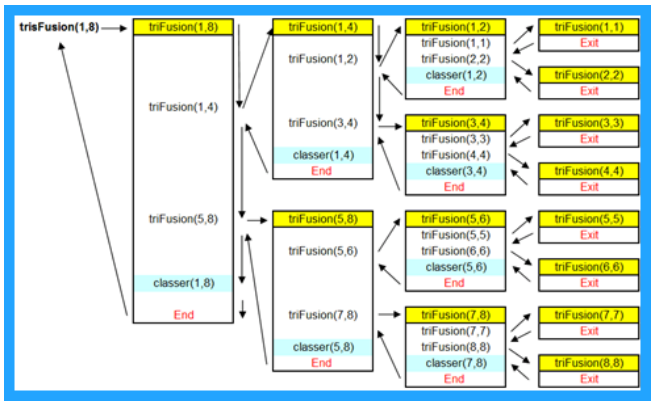
Tri	Tri par sélection	Tri à bulles
Principe	On **échange** chaque valeur (sauf la dernière) **contre le minimum** (maximum) **à partir** de cette valeur à échanger	Pour *n-1* **parcours** en partant de la fin, on échange chaque valeur qui n'est pas bien placée avec **sa voisine**. Après chaque parcours, on sait qu'au moins une valeur de plus au début est **bien placée**, et qu'on a déjà **terminé** le tri si on n'a fait **aucun échange** lors du parcours
Illustration	 **Performance** *(nombre approximatif d'opérations pour trier n valeurs) :* **Performance constante** : pour **chaque** valeur à échanger, il faut comparer **chaque** valeur → *n * n* donc ***n²* opérations**	 **Performance** *(nombre approximatif d'opérations pour trier n valeurs) :* **Pire scénario** : chaque valeur est mal placée. On fait à peu près *n* **parcours** avec *n* **comparaisons** par parcours → ***n²* opérations** **Meilleur scénario** : chaque valeur est dès le départ bien triée. **Un seul parcours** de *n* comparaisons suffit pour voir qu'on ne fait **aucun échange** → ***n* opérations** si la liste est déjà triée au départ
Programmation	**Pour chaque échange**: Pour chaque ligne ***nl1*** (sauf la dernière) La position du minimum *minNl* commence à *nl1* **Pour chaque comparaison** : pour chaque ligne ***nl2*** après la ligne *nl1* **Si** la valeur à *nl2* est inférieure à celle à *minNl* C'est *nl2* qui devient le nouveau *minNl* **Comparaison suivante** : *nl2* + 1 **Echange** des valeurs entre la ligne nl1 et la ligne minNl **Echange suivant** : *nl1* + 1	**Pour chaque parcours (inversé)**: Pour chaque ligne ***i*** (sauf la dernière) On baisse un **drapeau** (variable *échange*) **Pour chaque comparaison** 2 à 2: pour chaque ligne ***nl*** à partir de l'avant-dernière jusqu'à la ligne i **Si** la valeur ligne *nl* est supérieure à celle à *nl*+1 **Echange** des valeurs entre la ligne *nl* et la ligne *nl*+1 On lève le **drapeau echange** **Comparaison suivante** (donc valeur précédente) : *nl* - 1 Si le drapeau *échange* est toujours baissé : on arrête le tri **Parcours suivant** : *i* + 1
Tri décroissant	*minNl*→***maxNl***, qui change quand la valeur *nl2* est **>** à la m*axNl*	On échange cette fois quand la valeur *nl* est plus **petite** que la *nl*+1

Tri-fusion *(**performance :** n * log₂(n) car on minimise les comparaisons)*



Exemple avec 16 valeurs : on **divise** la liste de valeurs en **2 moitiés récursivement** jusqu'à n'avoir que des **valeurs uniques** qui sont groupées 2 par 2 en **8 paquets de 2 valeurs ordonnées**, qui sont eux fusionnés (et ordonnés) 2 par 2 en **4 paquets de 4 valeurs**, qui vont devenir **2 paquets de 8 valeurs ordonnées**, puis **1 paquet de 16 valeur ordonnée**. Le tri-fusion oblige en fait à **copier dans une colonne à droite** le paquet à diviser en 2 moitiés, puis **supprimer** cette colonne à droite quand on aura **fusionné** les 2 moitiés dans la colonne de gauche.

Logique : **Double appels récursifs** (sur **chaque moitié** de la plage) : **à l'aller**, on **copie** la plage dans la colonne de **droite**, et on exécute **l'appel récursif sur chaque moitié** de la **colonne de droite**.
Au retour, on exécute une **autre procédure** qui **classe les 2 paquets de la colonne en 1 seul paquet** ordonné dans la colonne de **gauche**, puis on **supprime la colonne**.
Conseil : le tri-fusion ne peut généralement bien se comprendre qu'en exécutant **les exemples de la SL8 dans EasyVBA**



Programmation :

- **Pas de boucle**, mais une **double récursion** (sur **chaque moitié** de la plage copiée dans la **colonne de droite**)
- **Avant la double récursion** : **arrêt de la récursion** si **1 seule ligne** à trier, puis **copie** de la plage dans la **colonne de droite**, calcul de la **position du milieu** de la colonne de droite
- **2 Appels récussifs : sur chaque moitié** dans dans la **colonne de droite**
- **Après la double récursion** : appel de la procédure **d'interclassement**, puis suppression de la colonne (car la procédure d'interclassement a fait le classement/fusion dans la colonne à gauche)

Interclassement **Paramètres** : la colonne, et les débuts et fins des 2 paquets (*d1,fin1,d2,fin2*).
Initialiser les 2 **positions** (dans les paquets) *p1* et *p2* à *d1* et *d2*, et la position *p* dans colonne de gauche à *d1*
Tant qu'il y a des valeurs dans les 2 paquets, ajouter la plus **petite** valeur entre la *p1* (qu'on augmente alors) et la valeur à *p2* (qu'on augmente alors) **à la position *p* dans la colonne de gauche** (*p* augmente à chaque tour).
Quand un **paquet est fini** (*p1>fin1* ou *p2>fin2*), on **copie le reste** de l'autre paquet à la fin de la colonne de gauche