BOUCLE FOR-END
FONCTIONS

BOUCLES FOR-END :

Il est possible de programmer une boucle qui se répètera un nombre donné de fois. Il s'agit de l'instruction :

```
for compteur = valeur_départ : valeur_finale
    Instruction1
    Instruction2
end
```

La variable compteur est initialisée à la valeur de départ et augmente d'une unité chaque fois que les instructions 1 et 2 ont été exécutées et ce jusqu'à la valeur finale.

1. Taper >> a = 3, for i = 0 : 6, a = a + i ; a, end

2. La valeur finale ne sera pas dépassée. Taper : >> for i = 1 : 4.5, i , end

Il est possible d'impose un pas différent de 1 au compteur :

```
for compteur = valeur_départ : pas : valeur_finale
```

3. Exemple avec la liste des nombres pairs : >> for i = 0 : 2 : 20, i,end

4. Le pas peut-être négatif. Taper : >> for i = 3 : -1 : -2, i,end

5. Avec une boucle for-end calculer la somme des entiers de 1 à 8.

6. Avec une boucle for-end calculer la somme des carrés des entiers de 1 à 8.

7. Avec une boucle for-end calculer 8 !

8. Soit la suite définie par $u_0 = 1$ et $\forall\, n \in \mathbb{N}$, $u_{n+1} = 2 \times u_n - 3$. Calculer u_8 puis u_{20}.

Conclusion : la boucle FOR-END s'utilise quand on connaît le nombre d'itérations.

FONCTIONS MATLAB :

Lorsque l'on souhaite que Matlab exécute plusieurs opérations successives, il est plus pratique de réaliser une fonction que de taper les instructions en ligne.

Une **fonction** est une succession d'instructions stockée dans un fichier fonction.m.

Pour créer une fonction il suffit d'aller dans le menu File, onglet New, cliquer sur M-File (ou cliquer sur l'icône New M-File).

La première ligne est impérativement de la forme : **function nom_de_la_fonction**.

Son nom de sauvegarde est obligatoirement nom_de_la_fonction.m (c'est d'ailleurs celui que propose Matlab par défaut).

Attention : le nom des fonctions ne doit pas contenir d'espace, ni d'accents, ni de caractères spéciaux : % @ ^ ( ) & , …

Voici un exemple simple :

```
function suite
u= 1 ;
for i = 1 : 20
   u = u*2 – 3 ;
end
```

Remarquer l'indentation proposée automatiquement par MATLAB. Elle facilite grandement l'analyse des instructions. Essayez de la conserver en cours de programmation.

MATLAB peut automatiquement remettre en place l'indentation avec la fonction Smart Indent dans le menu Text (Ctrl+I).

Pour utiliser cette fonction il suffit de taper son nom (sans le .m) sous MATLAB : >> suite.

Il est possible de transférer à la fonction des paramètres d'entrée et de récupérer des valeurs en sortie. Sa première ligne est alors de la forme :

function [variable_sortie_1, variable_sortie_2,…] = nom_de_la_fonction(variable_entrée_1,…).

Voici un exemple simple :

```
function [y] = suite2(n)
y= 1 ;
for i = 1 : n
   y = y*2 – 3 ;
end
```

n est le paramètre d'entrée (il indique le rang du terme de la suite qui sera calculé) et y est la variable de sortie (ici la valeur du $n^{ème}$ terme calculé).

En tapant >> [y] = suite2(4) on obtient le $4^{ème}$ terme de la suite puisque l'on fait appel à la fonction suite2 avec comme paramètre d'entrée la valeur 4. La fonction renvoie ici une seule valeur (y).

9. Taper [y] = suite2 et retenir la phrase d'erreur indiquée par Matlab.

Attention : lorsque l'opérateur fait appel à la fonction suite2 il doit forcément donner la valeur du paramètre d'entrée sinon la fonction ne peut pas s'exécuter.

EXERCICES :

10. Modifier la fonction suite2.m afin que l'opérateur saisisse u_0 en plus du rang n (suite3(u0,n)).

11. Créer une fonction permettant de calculer le $n^{ème}$ terme de la suite u_n définie par $u_0 = 2$ et $\forall n \in \mathbb{N}$, $u_{n+1} = u_n^2 - 1$. La valeur de n est fixée lors de l'appel à la fonction sous MATLAB.

12. Taper une fonction calculant le $n^{ième}$ et $(n-1)^{ème}$ termes de la suite de Fibonacci ($u_{n+2} = u_{n+1} + u_n$). L'opérateur choisit n, u_0 et u_1 (fibo(u0,u1,n)).

13. Taper une fonction [rep] = degre1(a,b) qui résout ax+b = 0 dans $\mathbb{R}$ où a et b sont deux réels.

La tester avec (2,3), (2,0), (0,2) et (0,0).

14. Taper une fonction [rep] = degre2(a,b,c) qui résout $ax^2+bx+c = 0$ dans $\mathbb{R}$ où a, b et c sont trois réels. Si a = 0, alors la fonction degre2 utilise la fonction degre1 pour déterminer rep. S'il existe deux racines alors rep est un vecteur a deux composantes : [rep] = [racine1, racine2].

La tester avec (2,3,1), (3,2,1), (1,2,1), (0,2,1) et (0,0,0).

Un entier est premier s'il est différent de 1 et s'il n'admet pas de diviseur autre que 1 et lui-même.

15. Écrire une fonction qui teste si un entier n donné est premier ([rep] = premier(x)).

16. Écrire une fonction qui affiche les nombres premiers inférieurs ou égaux à un entier saisi par l'opérateur (`[nb] = list_premier(x)` fera appel à la fonction `premier`). La fonction renvoie également en sortie le nombre `nb` de termes de nombres premiers inférieurs ou égaux à x.

COMPLEMENTS :

Les variables d'une fonction sont LOCALES : lors de l'appel à une fonction Matlab ouvre une espace mémoire vierge dans lequel aucune des variables du programme principal ne sont connues.
Si une variable du programme principal doit être utilisée dans une fonction il faut la mettre dans la liste des variables d'entrée.

17. Il est impossible de lancer la fonction `suite2` sans lui indiquer la valeur de la variable d'entrée (`n`) même si `n` est défini dans la mémoire de Matlab : durant le calcul de `suite2`, Matlab n'a plus accès à la mémoire générale.

Les variables de la fonction étant locales, il n'est pas nécessaire que les paramètres d'entrée aient le même nom dans la fonction et dans l'appel à la fonction. Ainsi la saisie suivante fonctionne tout à fait correctement : `>> rang = 4; [y] = suite2(rang)`. Et ce même si la fonction `suite2` stocke la valeur du paramètre d'entrée dans la variable `n` au lieu de `rang`.

Le fait que le nombre d'itérations s'appelle `rang` lors de l'appel à la fonction et `n` dans la fonction ne pose pas de problème : l'information passée à la fonction est la valeur numérique de la variable `rang` qu'elle stocke dans `n`.

De même si la variable de sortie ne porte pas forcément le même nom dans la fonction et dans l'appel à la fonction : `>> [resultat] = suite2(rang)` donne bien le bon résultat car la fonction `suite2` renvoie la valeur numérique de la variable qu'elle a calculé.

Ce que l'on retient :

- l'utilisation de la boucle **`for`-`end`** (nombre connu d'itérations).
- la syntaxe des fonctions (`function [var_s_1, var_s_2] = nom_de_la_fonction(var_e_1,var_e_2)`),
- l'indentation pour faciliter la lecture,
- complément : les variables des fonctions sont **locales** (signification et conséquences).