1. Introduction

Pour définir une fonction (ou procédures) en Scilab, la méthode la plus courante est de l'écrire dans un fichier, dans lequel on pourra d'ailleurs mettre plusieurs fonctions (en regroupant par exemple les fonctions qui correspondent à un même thème ou une même application).

2. Syntaxe d'une fonction

function $[y_1, y_2, \ldots y_n] = nomfonction(x_1, x_2, \ldots, x_p)$

instructions de la fonction

endfunction

// Où les $x_1, x_2, \ldots, x_p$ sont les arguments ou paramètres d'entrée ;

// Les $y_1, y_2, \ldots, y_p$ étant les arguments ou les résultats de sortie

3. Remarques

- Une fonction peut avoir **0, 1 ou plusieurs paramètres** (placés entre parenthèses)
- Ainsi que **0, 1 ou plusieurs résultats** (placés entre crochets).
- La tradition est de suffixer les noms des fichiers contenant les fonctions en **.sci**

4. Dans la pratique :

- Utilisez un éditeur de texte pour définir votre fichier de fonctions que vous enregistrerez dans votre répertoire de travail (ou un répertoire de fonctions)
- Dans Scilab, vous devez lui indiquer où se trouve les fonctions. Il faut alors charger le fichier avec l'instruction **exec** en lui indiquant le chemin et/ou le fichier en question.
- Vous pouvez alors utiliser les noms de fonctions comme des mots clés supplémentaires du langage Scilab.

Exemple

Construction du script de la fonction factorielle dans un éditeur de texte que l'on enregistre dans le fichier fact.sci :

```
function [y]=factorielle(n)
  y=1
  for i=1:n do
    y=y*i
  end
endfunction
```

Puis, par exemple, dans la console Scilab (si vous avez déclaré le bon répertoire de travail) :

```
--> exec('fact.sci')
--> disp(factorielle(10))
    3628800
```

5. Rôles des variables

- Dans l'écriture des fonctions, l'argument d'entrée x et l'argument de sortie y sont des variables ou arguments formels. Alors que dans son utilisation :resultat =factorielle (10), les arguments utilisés sont appelés arguments effectifs.
- L'argument effectif d'entrée peut être :
  - Une constante,
  - Une variable,
  - Le résultat d'une expression.

- Dans une fonction, vous avez accès (en lecture uniquement) à toutes les variables des niveaux supérieurs. On peut alors utiliser les variables globales de programmes (si elles existent). Cela permet de ne pas consommer trop de mémoires (pour la recopie dans une autre variable d'entrée) et de temps (pour le temps de la recopie).
  Si vous tentez de modifier cette variable globale, une nouvelle variable interne (à la fonction) est créée, la variable de même nom du niveau supérieur n'est alors pas modifiée.

Exemple

```
function [y1]=test(x1)
  y1=x1+c   // on utilise la variable globale c (si elle existe …)
  disp(c,'c = ')  // disp permet de visualiser la valeur globale c
  c=rand()  // une variable interne c est créée
  disp(c,'c = ')  // affichage de la variable interne
endfunction
```

- Il n'est donc pas possible de modifier une variable d'un programme à partir d'une fonction.
- Lors de la sortie de la fonction, toutes les variables internes (donc propres à la fonction) sont détruites.

7. Instructions spécifiques

1) Débogage de fonctions (Elimination d'anomalies)

- On peut dans un premier temps utiliser la fonction disp(v1,v2, …) qui permet de visualiser l'évolution de certaines variables de la fonction.
- On peut également mettre une ou plusieurs instructions **pause** en des endroits stratégiques de la fonction. Lorsque Scilab rencontre cette instruction, le déroulement du programme s'arrête et vous pouvez examiner la valeur de toutes les variables déjà définies (l'invite --> se transforme en -1->). Lorsque vos observations sont finies, la commande **resume** fait repartir le déroulement des instructions (jusqu'à l'éventuelle prochaine **pause**).

La fonction suivante mis en évidence le principe de l'instruction **pause**, après exécution de **pause on peut vérifier la valeur de «** a » et on peut aussi la changer si nécessaire :

Exemple :

```
function c=foo(b)
        a=42+b
        disp("Merci pour la  pause, vous pouvez vérifier si  "a" a la bonne  valeur !");
        disp("La valeur peut aussi être  changée au besoin !");
        pause
        c=a+2
endfunction
```

2) Interruption d'une procédure

L'instruction **return** permet d'interrompre le déroulement d'une fonction et de revenir au programme ayant fait appel à cette fonction. Si la fonction a été déclarée comme fournissant un résultat, la (ou les variables) constituant ce résultat doivent avoir reçu préalablement une valeur.

Exemple :
La fonction suivante fournit l'inverse de son paramètre, après avoir testé que celui-ci n'est pas nul ; dans ce dernier cas, elle rend la valeur infinie :

```
function [z]=inv(x)
    if x == 0 then
       z=%inf;
      return
   end
   z = 1/x
endfunction
```

8. Exemple de fonction renvoyant deux arguments

a) Résolution d'une équation du second degré avec $\Delta > 0$ et a<>0

```
function [x1, x2]=resol(a, b, c)
        if a == 0
           disp('Erreur !on ne traite pas le cas a = 0')
        else
          delta=b^2-4*a*c
          if delta < 0
            disp (" Erreur ! on ne traite que le cas où delta >0 !")
            x1='erreur delta est négatif'
          else
          x1 =(-b-sqrt(delta))/(2* a);
          x2 =(-b+sqrt(delta))/(2* a);
         end
        end
endfunction
```

Attention, il y a deux valeurs résultats, si on exécute dans Scilab : --> resol(1,-6,5) cela donne :

```
ans  =    1.
```

Le résultat est mis dans **ans** mais **ans** est seule pour récupérer les deux résultats, par défaut, elle prend le premier des deux résultats. Pour récupérer les deux résultats, il faut utiliser la syntaxe :

```
--> [x1,x2]=resol(1,-6,5)
 x2  =
   5.
 x1  =
   1.
```

b) Généralisation : cette fonction résout une équation du second degré dans le cas général

```
// Résolution de l'équation a*x*x+b*x+c = 0 dans le cas général
// syntaxe [r1,r2]= resol2(a,b,c)
// Entrées : a,b,c = coefficients de l'équation
// Sorties : r1,r2 = les deux racines de l'équation
function [r1, r2]=resol2(a, b, c)
  delta = b*b-4*c;
  r1=(-b-sqrt(delta))/2*a;
  r2=(-b+sqrt(delta))/2*a;
endfunction
```

Réalisation pratique et exécution

- Créez ce fichier avec l'éditeur de SCILAB.
- La fonction doit d'abord être chargée dans SCILAB avant d'être utilisée :
  - À partir de la fenêtre d'édition (ctrl-L),
  - Ou dans la fenêtre de commandes à l'aide de l'instruction : exec("resolequ2.sci");
- Une fois chargée, la fonction peut être utilisée comme une fonction prédéfinie de Scilab

Exemple d'exécution :

```
--> [r1,r2]=resol2(1,-6,5)     // deux racines distinctes
 r2  =
   5.
 r1  =
   1.
 --> [r1,r2]= resol2(1,-6,9)            // racine double
 r2  =
   3.
 r1  =
   3.
--> [r1,r2]= resol2(1,0,4)             // racines complexes
 r2  =
   2.i
 r1  =
  -2.i
```