Les fonctions

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). Chaque fonction doit commencer par l'instruction :

function [y1,y2,…,yn]=nomfonction(x1,x2,…,xp)

où les xi sont les arguments d'entrée (ou paramètres), les yj étant les arguments de sortie (ou résultats). Une fonction peut avoir 0, 1 ou plusieurs paramètres (placés entre parenthèses) ainsi que 0, 1 ou plusieurs résultats (placés entre crochets).

Il n'y a pas de mot clé délimitant la fin d'une fonction (comme end ou end function). Dans le cas où le fichier contient plusieurs fonctions, ce rôle est joué par chaque déclaration de fonction via le mot clé function ou par la fin du fichier pour la dernière fonction (dernier saut de ligne).

Remarques :
- La première ligne du fichier de fonctions doit commencer par l'instruction function
- La dernière instruction doit obligatoirement être suivie d'un passage à la ligne sinon l'interpréteur ne la prend pas en compte.
- La tradition est de suffixer les noms des fichiers contenant les fonctions en .sci (en salle informatique, avec le bloc-note, nous le laisserons le suffixer en fichier texte c'est-à-dire en .txt)

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 getf 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 ou fact.txt) :

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

Puis, dans Scilab (si vous avez déclaré le bon répertoire de travail avec Change Directory) :

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

Rôles des variables

Dans l'écriture des fonctions, l'argument d'entrée n 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.
Le (ou les) 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 des variables dans l'ordre contraire de la déclaration
c=rand()  // une variable interne c est créée
disp(c,'c = ')  // affichage de la variable interne
```

Passage par paramètres

Les premières versions de Scilab donnaient les variables d'entrée par valeur, c'est-à-dire que seules les valeurs des variables étaient affectées aux arguments formels. Une modification de ces arguments formels n'avait aucune incidence sur les variables initiales.
A partir de la version 2.4 de Scilab, les variables d'entrées sont passées par référence (ou adresse). Ce n'est plus la valeur mais la référence de la variable qui est donnée et ceci afin de diminuer la place en mémoire des définitions de variables. Par contre, dès que cette variable est susceptible d'être modifiée, une copie en est donnée et la variable initiale n'est pas modifiée.
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.

Instructions spécifiques

Déboguage de fonctions

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).

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 infini :

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

Exemple d'une fonction renvoyant deux arguments : Résolution d'une équation du second degré avec $\Delta > 0$

```
Function [x1,x2] = resol(a,b,c)
if (a==0) then error("on ne traite pas le cas a=0!")
else
   delta   = b^2-4*a*c
   if (delta <0) then error("on ne traite que le cas où delta > 0")
   else if (b<0) then x1=(-b-sqrt(delta))/(2*a);x2=(-b+sqrt(delta))/(2*a);
                      end
  end
end
```

Attention, il y a deux valeurs, si on exécute dans Scilab resol(1.e-08,0.8,1.e-08)

```
ans =
         -1.250e-08
```

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

```
--> [x1,x2]=resol(1.e-08,0.8,1.e-08)
x2 =
  - 800000000.
x1 =
  - 1.250e-08
```