Introduction à Scilab (3)

5 - Programmation

Scilab dispose d'un langage avec instructions structurées afin d'écrire des programmes complexes. Le langage est de type *impératif* (comme par exemple les langages C, Java, Ada).

5.1 - Variables et affectation

On peut stocker différentes valeurs (ou tableaux de valeurs) dans des variables, une variable étant identifiée par un nom, nom formé d'une lettre suivie éventuellement de lettre(s) et/ou de chiffre(s).

En Scilab, une variable se définit *à la volée* en lui affectant le résultat d'une expression.

Affectation

Notation algorithmique	Notation Scilab
variable ← *expression*	*variable* = *expression*

5.2 - Entrée-sortie

L'instruction `disp` permet l'affichage d'une variable ou d'une expression.
L'instruction `input` permet à l'utilisateur d'entrer une valeur, un tableau ou une chaîne de caractères.
Exemple : créez le fichier suivant en le nommant `ex_entree_sortie.sce`

```
n = input("Entrer un entier : ")
mprintf("n   = %d\n",n)
mprintf("n*n = %d\n",n*n)

t = input("Entrer un tableau de valeurs (entre crochets) : ")
disp(t)
```

puis exécutez le script `ex_entree_sortie.sce`.
⇨ dans certains cas, à la place de l'instruction `disp`, on utilisera l'instruction `mprintf` (correspondant à l'instruction `printf` du langage C) pour écrire des nombres (entiers ou réels).

Entrée-sortie

Notation algorithmique	Notation Scilab
lire (*variable*)	`input(`*variable*`)` `input(`*texte*`,`*variable*`)`

ecrire (*variable*)
ecrire (*valeur_entiere*)
ecrire (*valeur_reelle*)
	`disp(`*variable*`)`
	`mprintf("%d\n",`*valeur_entiere*`)`
	`mprintf("%f\n",`*valeur_reelle*`)`

5.3 - Test

L'instruction `if` permet d'exécuter une suite d'instructions si et seulement si une expression est vraie.

Test simple

Notation algorithmique	Notation Scilab

si *expression_booleenne* **alors**
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`end`

si *expression_booleenne* **alors**
instructions
sinon
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`else`
	instructions
	`end`

On peut aussi enchâiner plusieurs tests à la suite :

Test multiple

Notation algorithmique	Notation Scilab

si *expression_booleenne* **alors**
instructions
sinon_si *expression_booleenne* **alors**
instructions
⋮
sinon
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`elseif` *expression_booleenne* `then`
	instructions
	⋮
	`else`
	instructions
	`end`

Exemple : créez le fichier suivant en le nommant `ex_test.sce`

```
n = input("Entrer un entier n : ")
disp(n)


if n>2 then
    disp("n est supérieur à 2")
end

if n==0 then
    disp("n est nul")
elseif n>0 then
    disp("n est strictement positif")
else
    disp("n est strictement négatif")
end
```

puis exécutez le script `ex_test.sce`.

➪ les opérateurs de comparaison sont :

`<` inférieur strictement à	`>` supérieur strictement à	`==` égal à
`<=` inférieur ou égal à	`>=` supérieur ou égal à	`<>` différent de

➪ les opérateurs booléens sont :

`&` ET	`|` OU	`~` NON

5.4 - Boucle conditionnelle

L'instruction `while` permet de répéter une suite d'instructions tant qu'une expression booléenne est vraie.

Boucle conditionnelle

Notation algorithmique	Notation Scilab
tant_que *expression_booleenne* **faire** *instructions* **fin_tant_que**	`while` *expression_booleenne* *instructions* `end`

Exemple : créez le fichier suivant en le nommant `ex_boucle1.sce`

```
// définir deux entiers a et b tels que 0 < a < b
a = 48; b = 66;
mprintf("a = %d , b = %d \n", a, b)
while a>0
    r = a;
    a = pmodulo(b,a);
    b = r;
end
mprintf("le pcgd de a et b est %d\n", b)
```

puis exécutez le script `ex_boucle1.sce`.

5.5 - Boucle inconditionnelle

L'instruction `for` permet de répéter une suite d'instructions pour un ensemble de valeurs.

Boucle inconditionnelle

Notation algorithmique	Notation Scilab
pour *variable* **appartenant à** *tableau_valeurs* **faire** *instructions* **fin_pour**	`for` *variable* = *tableau_valeurs* *instructions* `end`

Souvent, on l'utilise avec la syntaxe suivante :

Notation algorithmique	Notation Scilab
pour *variable* **de** *v_min* **à** *v_max* **faire** *instructions* **fin_pour**	`for` *variable* = *v_min*:*v_max* *instructions* `end`

et les instructions sont exécutées pour *variable* = *valeur_min*, *variable* = *valeur_min*+1, ..., jusqu'à *variable* = *valeur_max*

Exemple : créez le fichier suivant en le nommant `ex_boucle2.sce`

```
disp("Exemple 1")
for i = 1:10
    disp(i)
end

disp("Exemple 2")
tab_v = [2.4 7.4 8 3.1 9.5 0.1]
somme = 0;
for i = tab_v
```

```
    disp(i)
    somme = somme+i;
end
mprintf("La somme des elements de v est %f\n", somme);
```

puis exécutez le script `ex_boucle2.sce`.

5.6 - Commentaire

En Scilab, un commentaire correspond à la partie d'une ligne se trouvant après deux caractères `//` .

Exercice 1 :

écrire un script Scilab nommé `ex_suite.sce` et qui effectue les opérations suivantes :
- demande à l'utilisateur d'entrer une valeur positive a
- calcule le vecteur u formé de 10 valeurs de la manière suivante :

$$u(1) = \frac{a+1}{2} \quad \text{et} \quad u(n) = \frac{a}{2\,u(n-1)} + \frac{u(n-1)}{2} \text{ pour } n \text{ variant de 2 à 10}$$

- affiche les valeurs du vecteur u ainsi que les valeurs au carré du vecteur u.

5.7 - Procédure

Une *procédure* en Scilab généralise le concept de fonction mathématique, c'est à dire à partir d'un ensemble (éventuellement vide) de données, on exécute une suite d'instructions qui fournit un ensemble (éventuellement vide) de résultats.
On peut le schématiser ainsi :

En Scilab les ensembles de données / de résultats seront décrits par des listes de variables.

Sous-programme SCILAB

Notation Scilab

`function` [*liste_variables_resultats*] `=` *nom_sous_programme* `(`*liste_variables_donnees*`)`
instructions
`endfunction`

Exemple : avec l'éditeur de texte, créez le fichier suivant nommé `ex_statistiques.sce`, où une fonction calculant la moyenne et l'écart-type d'un tableau de valeurs est définie puis utilisée.

```
// fonction calculant la moyenne m et l'écart-type s
// d'un tableau de valeurs tv
function [m,s] = statistiques(tv)

  n = length(tv); // nombre d'éléments dans le tableau tv
  s1 = sum(tv);   // somme des éléments de tv
  s2 = sum(tv .^ 2);   // somme des éléments au carré de tv

  m = s1/n; // la moyenne
  s = sqrt(s2/n-m*m); // l'écart-type

endfunction
```

```
// calcul des statistiques pour le tableau t1
t1 = [1 5 7 2 3 1 3 2 7];
disp(t1);
[m1,s1] = statistiques(t1);
mprintf("moyenne = %f , ecart-type = %f\n", m1, s1);

// calcul des statistiques pour le tableau t2
t2 = input("Entrer un tableau de valeurs entre crochets : ");
[m2,s2] = statistiques(t2);
mprintf("moyenne = %f , ecart-type = %f\n", m2, s2);
```

Exemple : dans cet exemple, on va définir la fonction suivante

$$g(x) = \begin{cases} \exp(x) & \text{si } x < 0 \\ -x^2 + x + 1 & \text{si } x \geq 0 \end{cases}$$

puis l'évaluer pour différentes valeurs entre -1 et 1.
La fonction g est définie *par morceaux*, l'expression $y = g(x)$ dépend de l'intervalle auquel appartient x. Pour une telle fonction, on ne peut pas l'évaluer directement avec un tableau de valeurs, mais valeur par valeur.
Avec l'éditeur de texte, créez le fichier suivant nommé `ex_fonction_par_morceaux.sce` puis exécutez-le.

```
// définition de la fonction g(x)
function y=g(x)
        if x<0 then
                y = exp(x);
        else
                y = -x^2+x+1
        end
endfunction

// calcul et affichage de la fonction g
// pour les valeurs -1, -0.9, -0.8, ..., 0.9, 1
x = -1:0.1:1;
n = length(x); // nb de valeurs pour x
y = zeros(1,n); // créer un vecteur de meme taille que x
for i=1:n
        y(i) = g(x(i));
        mprintf("x = %f , y = %f\n", x(i), y(i));
end
```