



# scilab

## V - Boucles « tant que »

Lycée

Auteur : Raymond Moché

**Mots-clefs** : boucles « pour » et « while », instruction conditionnelle avec alternative, « floor », « trier », « prod », « factorielle », « format », « exec », « input », « reste », « quotient ».

### Références :

✓ Livret de présentation de « scilab pour les lycées » (2010) et Mise à jour et compléments (mars 2011) :

<http://www.scilab.org/education/lycee/docs>

✓ Aide scilab 5.3.2

[http://help.scilab.org/docs/5.3.2/fr\\_FR/index.html](http://help.scilab.org/docs/5.3.2/fr_FR/index.html)

### Syntaxe des boucles « tant que »

```
while conditions
  instructions ;
end
```

Tant que les conditions sont vérifiées, les instructions sont répétées.

### Liste des exercices :

**Énoncé n° 1** : [\*] Conversion du système décimal vers le système à base 2.

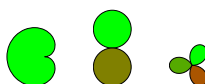
**Énoncé n° 2** : [\*] Sommes de nombres impairs.

**Énoncé n° 3** : [\*] Partie entière d'un nombre réel.

**Énoncé n° 4** : [\*] Formule de la somme des carrés.

**Énoncé n° 5** : [\*\*] Nombres factoriels.

**Énoncé n° 6** : [\*\*\*] Ranger 3 nombres donnés dans l'ordre croissant.



---

**Énoncé 1** [\*] Conversion du système décimal vers le système à base 2.

Un nombre entier  $n > 0$  est donné sous forme décimale. Le problème posé est de l'écrire dans le système à base 2, c'est à dire sous la forme d'une liste  $[a_0, a_1, \dots, a_p]$  de zéros ou de uns de sorte que  $n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + \dots + a_p \cdot 2^p$ .

---

**Mots-clefs** : boucle « tant que », commandes « input », « reste », « quotient ».

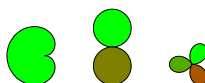
---

On sait que les coefficients  $a_0, \dots, a_p$  s'obtiennent par des divisions successives par 2, ce qui donne :

```
n=input('n='); // n est un entier >0.
A=[];
m=0;
c=0;
p=n;
while m<n
    a=reste(p,2);
    A=[A,a];
    m=m+a*2^c;
    p=quotient(p-a,2);
    c=c+1;
end
afficher(A,m)
```

La commande « input » permet de saisir la valeur de  $n$  dans la console. Dans l'exemple ci-dessous, on a choisi  $n = 123456789$ . Afficher  $m$  permet de vérifier le calcul car  $m$  est le résultat de la conversion de  $n$  du système à base 2 vers le système décimal. On revient au point de départ.

```
n=123456789
123456789.
column 1 to 12
1.  0.  1.  0.  1.  0.  0.  0.  1.  0.  1.  1.
column 13 to 24
0.  0.  1.  1.  1.  1.  0.  1.  1.  0.  1.  0.
column 25 to 27
1.  1.  1.
```



---

**Énoncé 2** [\*] Sommes de nombres impairs.

Calculer la somme des nombres impairs inférieurs ou égaux à un entier donné  $n$ .

---

**Mots-clefs** : boucle « tant que ».

---

Cet exercice est élémentaire. On remarquera l’affichage : il est constitué de 4 chaînes de caractères (deux chaînes limitées par le signe « ’ » au début et à la fin et 2 chaînes « string() »). Les 3 signes « + » concatènent ces 4 chaînes.

Listing 1 – Sommes de nombres impairs

```
// Somme S des nombres impairs <= n
n=input('n=');
S=0 ;// Initialisation de la somme
i=1;
while (2*i-1)<= n// début de la boucle.
    S=S+2*i-1;
    i=i+1;
end// Fin de la boucle.
afficher('La somme des impairs <= '+string(n)+' est '+string(S));
```

La commande « input » permet de saisir la valeur de  $n$ , ici 500, dans la console.

**Console Scilab**

```
-->exec('/Users/raymondmoche/Magasin_scilab/305/305_Somme-impairs.sce', -1)
n=500

La somme des impairs <= 500 est 62500

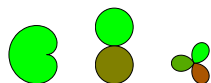
-->
```

En complément (ce n’est pas de l’algorithmique), on pourrait ensuite trouver la formule qui donne la somme  $S(n)$  des nombres impairs  $\leq n$  en remarquant que pour tout entier  $p \geq 0$

$$S(2p + 1) = (1 + 2 + \dots + (2p + 1)) - 2(1 + 2 + \dots + p)$$

D’après la formule qui donne la somme des  $m$  premiers entiers, si  $n = 2p + 1$ ,  $S(n) = \frac{(n + 1)^2}{2}$ . Enfin,

si  $n$  est pair,  $S(n) = S(n - 1) = \frac{n^2}{2}$ .



---

**Énoncé 3** [\*] Partie entière d'un nombre réel.

Calculer la partie entière d'un nombre réel donné.

---

**Mots-clefs** : instruction conditionnelle avec alternative, boucles « tant que », « floor ».

---

La partie entière d'un réel donné  $x$  est le plus grand entier  $n$  tel que  $n \leq x$ , autrement dit, l'unique entier  $n$  qui vérifie  $n \leq x < n + 1$ . Le script ci-dessous est très intéressant du point de vue de l'apprentissage de la programmation de calculs.

Listing 2 – Partie entière de  $x$

```
// Partie entiere d'un reel x
clear
x=input('x=');
y=0;//On va distinguer les cas x<0 et x>=0.
if y>x// Cas x<0.
    while y>x then
        y=y-1;
    end;
else// Cas x>=0.
    while y<=x then
        y=y+1;
    end
    y=y-1;
end;
afficher('La partie entière de x est '+string(y));
```

L'exécution de ce script pour trouver la partie entière de -2.117 donne :

Console Scilab

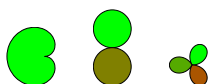
```
-->exec('/Users/raymondmoche/Magasin_scilab/305/305_Partie-entiere.sce', -1)
x=-2.117

La partie entière de x est -3

-->y=floor(x)
y =
- 3.

-->
```

On remarquera l'utilisation, à titre de vérification, de « floor », qui est justement la fonction « partie entière » préprogrammée dans « scilab ».



---

**Énoncé 4** [\*] Formule de la somme des carrés.

La formule  $1^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$  est-elle vraie pour tout entier  $n \geq 1$  ?

---

**Mots-clefs** : boucle « tant que », abandon d'un calcul.

---

Pour  $n = 1$ , cette formule est vraie. On augmente  $n$  de 1 (et on fera cela tant qu'elle restera vraie), on calcule  $S$  et  $T$  et on les compare. Si cette formule est vraie pour tout entier, l'algorithme ci-dessous ne s'arrêtera pas ! Dans ce cas, il faudra arrêter le calcul à l'aide de « Contrôle>Abandonner ».

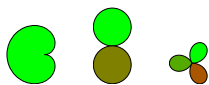
Lançons l'exécution de l'algorithme.

Listing 3 – La formule de la somme des carrés est-elle exacte ?

```
n=1;
S=1;
T=1;
while S=T
    n=n+1;
    S=S+n^2;
    T=n*(n+1)*(2*n+1)/6;
end
afficher(+string(n)+' est le plus petit entier pour lequel la formule
proposée est fausse');
```

On obtient la réponse : « 208065 est le plus petit entier pour lequel la formule proposée est fausse » qui est fausse. Cela est dû au fait que la capacité de « scilab » à traiter des nombres entiers en tant que tels a été dépassée. Voilà donc un exemple où « scilab » nous induit en erreur.

Il est clair qu'un calculateur numérique ne peut pas être utilisé pour démontrer qu'une infinité d'égalités sont satisfaites. Par contre, tout logiciel de calcul formel conviendra.



---

**Énoncé 5** [\*\*] Nombres factoriels.

1 - Combien y a-t-il de nombres factoriels inférieurs ou égaux à  $10^6$ ,  $10^9$ ,  $10^{12}$ ,  $(17!)$  ?  
2 - 3 629 300 et 39 916 800 sont-ils des nombres factoriels ?

---

**Mots-clefs** : boucles « pour » et « tant que », instruction conditionnelle, « prod », « factorielle », « format », « exec ».

---

Les scripts ci-dessous, rédigés de manière un peu compacte, appellent des commentaires :

- ✓ Comme on demande 4 fois combien il y a de factoriels inférieurs ou égaux à un entier donné, on fabrique dans le premier script une fonction-scilab qui associe à tout entier  $p \geq 1$  le nombre  $n_f(p)$  de factoriels  $\leq p$ .
- ✓ On calcule  $p!$  par la commande « `prod([1 : p])` ». On aurait été plus lent avec une boucle « pour » ou plus rapide avec « `factorielle(p)` ».
- ✓ On a imposé un « format » afin que « scilab » n'affiche pas certains grands entiers comme  $17!$  en virgule flottante.
- ✓ La fin du deuxième script se base sur le fait que  $p$  est un nombre factoriel si  $n_f(p-1) \neq n_f(p)$  et n'en est pas un sinon.

Listing 4 – Fonction de comptage des factoriels

```
function n=nombfact(p)
C=1; // les valeurs de C seront les entiers successifs.
fact=1; // les valeurs de fact seront les nombres factoriels successifs
while fact<=p then
    C=C+1;
    fact=fact*C;
end
n=C-1;
endfunction
```

Listing 5 – Compter et reconnaître des factoriels

```
clear
format ('v',25);
P=[10^6,10^9,10^(12),prod ([1:17])];
afficher (P);
exec ('/Users/raymondmoche/Magasin_scilab/305/305-nombfact.sci');
for i=1:4
    NF=nombfact(P(1,i));
    afficher ('Le nombre de nombres factoriels <= ' + string (P(1,i))
    + ' est ' + string (NF));
end;
// l'entier donné n est-il un nombre factoriel ?
n=[3629200,39916800];
for i=1:2
    if (nombfact(n(1,i)-1)==nombfact(n(1,i))) then
        afficher (string (n(1,i)) + ' n ' est pas un nombre factoriel.');
```

```

else
    afficher(string(n(1,i))+ ' est un nombre factoriel ')
end;
end;

```

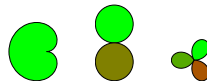
- ✓ La deuxième ligne de l'image ci-dessous montre que l'affichage de  $17!$  est satisfaisant. Néanmoins ces nombres suivis d'un « . » ne sont pas des entiers mais des réels pour « scilab ».
- ✓ Les nombres factoriels grandissant très vite, il est intéressant de voir jusqu'où on peut imposer un format acceptable. Sauf erreur, on touche aux limites de « scilab » et à la manière dont il calcule.

#### Console Scilab

```

-->exec('/Users/raymondmoche/Magasin_scilab/305/305-nombresfactoriels.sce', -1)
    1000000.    10000000000.    100000000000000.    355687428096000.
Le nombre de nombres factoriels <= 1000000 est 9
Le nombre de nombres factoriels <= 10000000000 est 12
Le nombre de nombres factoriels <= 100000000000000 est 14
Le nombre de nombres factoriels <= 355687428096000 est 17
3629200 n'est pas un nombre factoriel.
39916800 est un nombre factoriel
-->

```



---

**Énoncé 6** [\*\*\*] Ranger 3 nombres donnés dans l'ordre croissant.

Pour ranger 3 nombres donnés  $x$ ,  $y$  et  $z$  dans l'ordre croissant, on considère d'abord  $x$  et  $y$  que l'on échange si  $x > y$ ; puis on considère  $y$  et  $z$  et on fait de même; on compte aussi le nombre de permutations de 2 nombres consécutifs que l'on a réalisées au cours de ce premier passage. Si c'est 0, c'est terminé. Sinon, on fait un deuxième passage et ainsi de suite, éventuellement. On demande d'écrire la suite  $x$ ,  $y$ ,  $z$  dans l'ordre croissant et d'afficher le nombre de permutations faites.

**Mots-clefs** : instructions conditionnelles avec alternative, boucle « tant que », « trier ».

---

Listing 6 – Ranger 3 nombres dans l'ordre croissant.

```
x=input (" x=" )
y=input (" y=" )
z=input (" z=" )
S=1;
NP=0;
while S>0 then
  if x>y then
    S1=1;
    t=x;
    x=y;
    y=t;
  else
    S1=0;
  end;
  if y>z then
    S2=1;
    t=y;
    y=z;
    z=t;
  else
    S2=0;
  end;
  S=S1+S2;
  NP=NP+S;
end;
A=[x,y,z];
afficher (A);
afficher ('Le nombre de permutations effectuées est '+string(NP));
```

- ✓ L'intérêt du script ci-dessus est qu'on peut le modifier facilement afin de trier dans l'ordre croissant une suite de nombres réels de longueur quelconque (une suite de 2 ou 3 nombres se range très facilement dans l'ordre croissant). À partir de 4, c'est beaucoup plus compliqué à programmer. Le tri utilisé ici s'appelle « tri à bulle ».
- ✓ Il existe d'autres types de tri.

- ✓ Bien sûr, la fonction « trier » de « scilab » trie les suites, avec diverses options, voir l'aide en ligne ou un manuel.
- ✓ Ci-dessous, nous avons recommencé le tri à l'aide de « trier ».

```

Console Scilab
-->exec('/Users/raymondmoche/Magasin_scilab/305/305-rangerxyz.sce', -1)
x=-1.17;
y=-2.34;
z=-4.71;

- 4.71 - 2.34 - 1.17

Le nombre de permutations effectuées est 3

-->trier([-1.17,-2.34,-4.71])
ans =

- 4.71 - 2.34 - 1.17

-->

```

**Pour aller plus loin** (Calcul des probabilités) :

On peut imaginer que les nombres  $x$ ,  $y$  et  $z$  sont tirés au hasard dans l'intervalle  $[0, 1]$ .  $NP$  est alors une variable aléatoire.

**1** - Quelles valeurs  $NP$  peut-elle prendre ?

Pour compter les permutations, on peut imaginer que l'on a tiré 1, 2 et 3 (seules interviennent les inégalités), ce qui donne le tableau :

			$NP$
1	2	3	0
1	3	2	1
2	1	3	1
2	3	1	2
3	1	2	2
3	2	1	3

**2** - Avec quelles probabilités ces valeurs sont-elles prises ?

De simples considérations de symétrie indiquent que la loi de  $NP$  est

$NP$	0	1	2	3
probabilités	1/6	1/3	1/3	1/6

Plus précisément, le volume de  $((x, y, z); 0 < x < y < z < 1)$  est  $1/6$  parce qu'il y a 6 tétraèdres de ce type qui remplissent le cube unité (à des ensembles de volume nul près) et que ces tétraèdres ont le même volume par raison de symétrie.

