Université des Sciences et Technologies de Lille
U.F.R. de Mathématiques Pures et Appliquées

Agrégation externe **Année 2002-2003**

Compléments de documentation Scilab : affichage de texte et formatage de nombres

Il y a différents moyens de faire écrire du texte par Scilab. Nous allons en décrire trois : `disp`, `write` et `printf`. Le choix de l'une de ces fonctions dépend de la complexité du texte à afficher : est-ce un simple bout de phrase, ou contient-il des variables à remplacer par leur valeur ? Le choix dépend aussi des exigences que l'on peut avoir en termes de formatage (des nombres et du texte).

1 Le formatage des nombres affichés par Scilab

Commençons par une petite expérience, à réaliser au démarrage de Scilab, sans avoir touché à la fonction `format`

```
-->a=[0.123456789, 123.456789; -0.123456789, -123.456789]
 a  =

!   0.1234568    123.45679 !
! - 0.1234568  - 123.45679 !
```

Il semble que le comportement par défaut de Scilab soit l'affichage de 8 chiffres, avec arrondi si on lui en fournit plus en entrée. Si cet affichage décimal ne lui permet pas d'aller jusqu'au point décimal (équivalent de la virgule en notation française), il y a commutation automatique en mode exponentiel (avec puissances de 10) encore appelé mode « ingénieur » (en anglais exponential ou engineer, d'où la lettre E) :

```
-->b=[12345678  123456789]
 b  =

!   12345678.    1.235E+08 !
```

où l'écriture `1.235E+08` s'interprète comme $1{,}235 \times 10^8$.

La fonction Scilab qui contrôle l'affichage des nombres s'appelle `format`, regardons sa documentation en ligne.

```
NAME
   format - number printing and display format
```

```
CALLING SEQUENCE
 format([type],[long])
 format()
PARAMETERS
 type       : character string

 long       : integer ( max number of digits (default 10))

DESCRIPTION
   Sets the current printing format with the parameter type ; it is one of
  the following :

 "v"  : for a variable format (default)

 "e"  : for the  e-format.

     long defines the max number of digits (default 10). format() returns
      a vector for the current format: first component is the type of
      format (0 if v ; 1 if e ); second component is the number of digits.
```

On peut donc utiliser `format` avec 0, 1 ou 2 arguments. Avec zéro argument, la fonction format retourne le formatage en vigueur au moment de son appel :

```
-->format()
 ans  =

!   1.    10. !
```

nous indique que le type de formatage est `'v'`, c'est-à-dire variable[1] (c'est le format utilisé pour l'affichage du vecteur `b` ci-dessus) et que le nombre maximal de chiffres décimaux utilisés pour l'affichage des nombres est 10. Cette affirmation semble contredire l'observation faite sur la matrice `a` d'un affichage par défaut de seulement 8 chiffres décimaux. L'examen attentif des alignements verticaux dans l'écriture de la matrice `a` permet de subodorer la solution de l'énigme. On constate que les nombres négatifs de la deuxième ligne occupent bien 10 caractères : le signe moins, une espace[2] et les 8 chiffres décimaux, le point ne comptant pas. L'alignement vertical permet de voir que les nombres positifs de la première ligne sont écrits en commençant par deux espaces. Ils occupent donc eux aussi 10 caractères. Si cette conjecture est vraie, on devrait pouvoir afficher tous les chiffres des éléments de `a` en imposant un formatage sur 12 caractères puisqu'au départ nous avons saisi `a` avec des nombres à 9 chiffres et des nombres à 10 chiffres. Allons y progressivement :

1. Il y a manifestement une erreur dans la documentation en ligne, `'v'` correspond à 1 et `'e'` à 0 ; pour le voir, taper successivement `format('v');format()` et `format('e');format()`.
2. En typographie, « espace » est féminin.

```
-->format(11);a
 a  =

!   0.12345679    123.456789 !
! - 0.12345679  - 123.456789 !

-->format(12);a
 a  =

!   0.123456789    123.456789 !
! - 0.123456789  - 123.456789 !

-->format(13);a
 a  =

!   0.123456789    123.456789 !
! - 0.123456789  - 123.456789 !
-->format()
 ans  =

!   1.    13. !
```

Pour revenir au formatage par defaut, il suffit de taper `format('v',10)`.

Deux remarques pour finir : `format` permet de contrôler le nombre total de chiffres décimaux *affichés*, mais pas d'imposer le nombre de chiffres après la virgule ni de changer la précision des calculs qui dépend de la représentation binaire interne des nombres par Scilab [3].

```
-->format(20);a
 a  =

!   0.123456789    123.456789000000001 !
! - 0.123456789  - 123.456789000000001 !
-->format(26);a
 a  =

!   0.12345678899999999733605    123.456789000000000555701 !
! - 0.12345678899999999733605  - 123.456789000000000555701 !
```

Le résultat à première vue surprenant de ces deux dernières commandes ne devrait plus l'être si l'on se souvient que tous les nombres représentés en machine le sont sous forme binaire. Par conséquent lorsque l'on saisit au clavier le rationnel décimal $r =$

3. Qui n'a aucune raison d'être affectée par la commande `format`, cette dernière n'intervenant qu'au niveau de l'affichage.

0.123456789, il est stocké en mémoire par Scilab sous la forme d'une approximation *dyadique* $k2^{-n}$ avec k et n entiers. Si l'on augmente le format d'affichage en passant à 20 ou 26 chiffres décimaux, Scilab affiche une approximation *décimale* de $k2^{-n}$ et non pas de r.

2 Affichage rudimentaire avec `disp`

D'après la documentation en ligne, la fonction `disp` permet l'affichage de variables de toute nature (nombres, matrices, texte) :

```
NAME
   disp - displays variables

CALLING SEQUENCE
 disp(x1,[x2,...xn])
DESCRIPTION
   displays xi with the current format. xi's are arbitrary objects (matrices
  of constants, strings, functions, lists, ...)
```

Les crochets désignent des arguments optionnels. Il faut donc toujours fournir au moins le premier argument x_1, les suivants étant facultatifs (et pas forcément de même nature que x_1). Voici une utilisation basique pour afficher du texte.

```
-->disp('Hypothèse rejetée')

 Hypothèse rejetée
```

Attention à ne pas confondre les crochets de la description syntaxique ci-dessus avec des délimiteurs de matrice. Les trois exemples suivants devraient lever l'ambiguïté :

```
-->a=3;disp(a,[1,2])

!   1.    2. !

    3.

-->disp(a,[2,'alors'])
            !--error     4
undefined variable : %s_c_c


-->disp(a,[2,3],'alors')

 alors
```

```
!   2.    3. !

    3.
```

Au passage, on aura remarqué les deux inconvénients de `disp` utilisé avec plusieurs arguments : l'inversion de l'ordre et le passage systématique à la ligne.

```
-->s1="Belle marquise";s2="vos beaux yeux";s3="me font";s4="mourir";

-->s5="d''amour";

-->disp(s1,s2,s3,s4,s5)

 d'amour

 mourir

 me font

 vos beaux yeux

 Belle marquise
```

On peut arranger cela en utilisant la concaténation des chaînes de caractères (opérateur `+`) :

```
-->s=s1+s2+s3+s4+s5
 s  =

 Belle marquisevos beaux yeuxme fontmourird'amour
```

Il faut bien sûr gérer à la main les espacements, soit en les incluant dès le départ dans les chaînes, soit en concaténant :

```
-->s=s1+' '+s2+' '+s3+' '+s4+' '+s5; disp(s)

 Belle marquise vos beaux yeux me font mourir d'amour
```

Ceci donne l'idée de contourner les inconvénients de `disp` en lui passant comme paramètre *une seule chaîne de caractères* éventuellement fabriquée par concaténation à partir de morceaux de phrases et de variables numériques, *à condition* que les valeurs de ces variables aient été converties en chaînes de caractères. Rappelons que ce travail est réalisé par la fonction `string`.

```
-->prixht=127.56;taux=0.216;string(prixht)
 ans  =
```

```
 127.56

-->string('prixht')
 ans  =

 prixht

-->p1='Le prix TTC est de ';p2=' euros dont '; p3=' euros de TVA';

-->disp(p1+string(prixht*(1+taux))+p2+string(taux*prix)+p3)

 Le prix TTC est de 155.11296 euros dont 27.55296 euros de TVA

-->taux=0.206; // baisse de la TVA

-->disp(p1+string(prixht*(1+taux))+p2+string(taux*prix)+p3)

 Le prix TTC est de 153.83736 euros dont 26.27736 euros de TVA
```

Cette méthode fonctionne assez bien pour un texte court (une ligne). Il reste un dernier problème : le formatage des nombres, on aimerait que les prix TTC et les montants de TVA soient affichés au centime d'euro près.

```
-->disp("Peut-on vraiment afficher ici un message plus long qu''une
ligne? Essayons pour voir.")
 Peut-on vraiment afficher ici un message plus long qu'une ligne? Essay
      ons pour voir.
```

3 Utilisation de `write`

Une autre façon de faire écrire du texte à Scilab est d'utiliser `write` qui permet d'écrire dans un fichier. Le fonctionnement est bien documenté dans le polycopié *Une introduction à Scilab* de B. PINÇON, pages 47–50. Pour le problème qui nous intéresse, il suffit de savoir que la « sortie standard » de Scilab est justement le fichier affiché par la fenêtre de Scilab. Autrement dit, écrire dans ce fichier revient à écrire sur l'écran. Les fichiers ouverts par Scilab sont repérés par un numéro entier. Pour voir les fichiers ouverts, il suffit de taper :

```
-->file()
 ans  =

!   1.    2.    5.    6. !
```

On voit ainsi qu'il y 4 fichiers ouverts (cela peut changer si vous essayez sur votre machine) portant les numéros 1, 2, 5 et 6. Pour en savoir un peu plus, il suffit de taper :

```
-->dispfiles()
|File name                |Unit|Type|Options           |
|-------------------------------------------------------|
|/home/suquet/scilab.hist|1   |F77 |unknown formatted |
|/tmp/SD_1393_/foo        |2   |F77 |unknown formatted |
|Input                    |5   |F77 |old formatted     |
|Output                   |6   |F77 |new formatted     |
```

On récupère ainsi quelques informations, dont le nom de chemin des fichiers. Les deux derniers sont un peu spéciaux, ce sont justement l'entrée standard (Input) et la sortie standard (Output). Vérifions :

```
-->write(6,"Coucou, c''est moi!")
Coucou, c'est moi!
```

Les numéros de ces deux fichiers Input et Output sont toujours mémorisés dans la variable vectorielle `%io` (pour **i**nput-**o**utput).

```
-->%io
 %io  =

!   5.    6. !
```

Le numéro attribué par Scilab au fichier de sortie standard est donc mémorisé dans `%io(2)`, deuxième composante du vecteur `%io`. Pour la portabilité, il est donc préférable de coder plutôt que `write(6,"Coucou, c''est moi !")`,

```
-->write(%io(2),"Coucou, c''est moi!")
Coucou, c'est moi!
```

Pour l'affichage d'une ligne de texte, on peut faire la même chose à quelques nuances près avec `disp` ou `write(%io(2),...)` :

```
-->write(%io(2),[s1,s2,s3,s4,s5])
Belle marquise
vos beaux yeux
me font
mourir
d'amour
-->write(%io(2),p1+string(prixht*(1+taux))+p2+string(taux*prix)+p3)
Le prix TTC est de 153.83736 euros dont 26.27736 euros de TVA

-->write(%io(2),"Peut-on vraiment afficher ici un message plus long q
u''une ligne? Essayons pour voir.")
Peut-on vraiment afficher ici un message plus long qu'une ligne? Essa
yons pour voir.
```

On peut écrire aussi dans un fichier des objets de nature différente, mais sur des lignes différentes en faisant à chaque fois un nouvel appel à `write`, cf. l'exemple présenté p. 49 dans le polycopié de B. PINÇON. On peut alors exploiter les possibilités de formatage des nombres.

4 Utilisation de `printf` et `mprintf`

Les fonctions `printf` et `mprintf` sont des émulations de fonctions C et offrent beaucoup plus de fonctionnalités que ce que nous avons vu jusqu'ici. La documentation en ligne de `printf` n'est pas très claire (à mon avis) par manque d'exemples. Elle est complétée par celle de `printf_conversion`.

La syntaxe générale de `printf` est la suivante :

```
printf('c1c2...cn',v1,v2,...,vn)
```

Sous cette forme, `printf` va afficher les valeurs des n objets `v1`,...,`vn`, qui peuvent être de nature différente (nombres réels, entiers ou chaînes de caractères), leur formatage étant contrôlé par le premier paramètre `'c1c2...cn'` qui est nécessairement une chaîne de caractères, la sous-chaîne `'c1'` donnant le formatage de `v1`, `'c2'` celui de `v2`, etc.

Sous sa forme minimale, la sous-chaîne de formatage `'ci'` controlant l'affichage de l'objet Scilab `vi` se compose des deux caractères `%` et une lettre indiquant le type de `vi`, les plus utiles (pour nous) étant `d` ou `i` pour un entier[4], `f` pour un *float*, en pratique un nombre décimal censé représenter un réel, `e` ou `E` pour la notation exponentielle ou « ingénieur » des nombres, `g` pour choisir automatiquement le mieux adapté entre `f` et `e` (et de même `G` pour `f` ou `E`) et enfin `s` pour une chaîne de caractères.

```
 -->printf('%s%f%s%f%s',p1,prixht*(1+taux),p2,prixht*taux,p3)
Le prix TTC est de 153.837360 euros dont 26.277360 euros de TVA
```

L'intérêt de `printf` est que l'on peut faire nettement mieux en choisissant la précision de l'affichage pour chacun des nombres :

```
-->printf('%s%.2f%s%.2f%s',p1,prixht*(1+taux),p2,prixht*taux,p3)
Le prix TTC est de 153.84 euros dont 26.28 euros de TVA

-->p3=p3+', ';p4='calculée au taux de ';
-->printf('%s%.2f%s%.2f%s%s%.3f',p1,prixht*(1+taux),p2,prixht*taux,p3,p4,taux)
Le prix TTC est de 153.84 euros dont 26.28 euros de TVA, calculée au taux de 0.206
```

Il est possible de forcer un passage à la ligne en insérant dans la chaîne de formatage la séquence `\n` (pour *new line*) :

```
-->printf('%s%.2f%s%.2f%s\n%s%.3f',p1,prixht*(1+taux),p2,prixht*taux,p3,p4,taux)
Le prix TTC est de 153.84 euros dont 26.28 euros de TVA,
calculée au taux de 0.206
```

4. Attention à `u` qui convertit en *unsigned integers*. Pour comprendre, essayez `printf('%u',n)` en donnant successivement à n les valeurs $2^{31}-1$, 2^{31}, $2^{31}+1$, -1, -2, $-2^{31}+1$.

Voici comment agit le paramètre optionnel précision sur le type `f` (*float*).
- Si ce paramètre est absent, `printf` affiche par défaut six chiffres décimaux après la virgule (le point).
- Si le paramètre s'écrit `.0`, `printf` affiche l'entier le plus proche écrit sans point décimal.
- Si le paramètre est un point suivi d'un nombre entier n (exemples `%.8f`, `%.15f`), `printf` affiche n chiffres après la virgule, en complétant avec des zéros si l'on est pas trop gourmand en précision, ou sinon d'une manière qui ressemble à ce que nous avons déjà observé avec `format` à la fin de la section 1.

```
-->y=12.347;

-->printf('\%f\n\%.0f\n\%.3f\n\%.15f\n\%.16f\n\%.26f',y,y,y,y,y,y)
12.347000
12
12.347
12.347000000000000
12.3469999999999995
12.34699999999999953104179440
```

Il y a un deuxième paramètre optionnel qui indique la largeur occupée par l'objet affiché en nombre de caractères. On le place avant le paramètre précision. Ainsi `%12.5f` indique un affichage de nombre *float* avec 5 chiffres après la virgule et occupant *au moins* 12 caractères (le point décimal compte ici pour un caractère, contrairement à ce qui se passe avec `format`). Si le nombre affiché n'a pas assez de chiffres pour occuper tout l'espace qui lui est attribué, l'affichage est complété avec des blancs. Le nombre affiché est par défaut tassé à droite de son espace réservé. Si on veut le tasser à gauche, il faut faire précéder la longueur de l'espace réservé du signe moins : `%-12.5f`. Si la précision indiquée ajoutée au nombre de chiffres avant la virgule dépasse la longueur de l'espace réservé, celui-ci est agrandi automatiquement. Par conséquent, `%5.12f` a le même effet que `%.12f`. Voici quelques exemples pour illustrer tout cela.

```
-->x=98.765432112345;

-->printf('%f%s%f',x,'poussez pas',x)
98.765432poussez pas98.765432

-->printf('%11f%s%11f',x,'poussez pas',x)
  98.765432poussez pas  98.765432

-->printf('%-11f%s%11f',x,'poussez pas',x)
98.765432  poussez pas  98.765432

-->printf('%-11f%.6s%11f',x,'poussez pas',x)
98.765432  pousse  98.765432
```

```
-->printf('%f\n%.11f\n%14f\n%-14f\n%14.8f\n%-14.8f',x,x,x,x,x,x)
98.765432
98.76543211234
     98.765432
98.765432
   98.76543211
98.76543211

-->printf('%.7f\n%8.7f\n%10.7f\n%11.7f\n%12.7f\n%-12.7f%s',x,x,x,x,x,x,'Stop')
98.7654321
98.7654321
98.7654321
 98.7654321
  98.7654321
98.7654321  Stop
```

On peut aussi faire opérer `printf` sur des matrices. Dans ce cas la chaîne de formatage opère ligne par ligne à travers toutes les matrices. Elle doit donc contenir autant de sous-chaînes de formatage qu'il y a de colonnes en tout et se terminer par un `\n`. Voici un exemple emprunté à (et adapté de) la documentation de `mprintf`.

```
-->couleurs=['rouge';'vert';'bleu';'rose';'noir'];

-->RGB=[1 0 0;0 1 0;0 0 1;1 0.75 0.75;0 0 0];

-->printf('%d%s%f%f%f\n',(1:5)',couleurs,RGB);
1rouge1.0000000.0000000.000000
2vert0.0000001.0000000.000000
3bleu0.0000000.0000001.000000
4rose1.0000000.7500000.750000
5noir0.0000000.0000000.000000
```

Sous cette forme ce n'est pas très lisible car on n'a pas pris soin de gérer les largeurs de colonnes. Voici une façon d'y remédier en fixant 3 chiffres après la virgule, des colonnes larges de 6 caractères (sauf la première à 3 caractères) et un alignement à gauche dans chaque colonne.

```
-->printf('%-3d%-6s%-6.3f%-6.3f%-6.3f\n',(1:5)',couleurs,RGB);
1  rouge 1.000 0.000 0.000
2  vert  0.000 1.000 0.000
3  bleu  0.000 0.000 1.000
4  rose  1.000 0.750 0.750
5  noir  0.000 0.000 0.000
```

On peut aussi construire des tableaux plus élaborés en enchaînant les instructions `printf`, comme ci-dessous où les trois lignes de commandes en forment en fait une seule grâce à l'utilisation de la séquence `...` (noter aussi l'usage des deux sauts de ligne dès le début pour la lisibilité).

```
-->printf('\n\n%-3s%-9s%-6s%-6s%-6s\n','no','couleur',...
-->'R','V','B');...
-->printf('%-3d%-9s%-6.3f%-6.3f%-6.3f\n',(1:5)',couleurs,RGB);


no couleur  R     V     B
1  rouge    1.000 0.000 0.000
2  vert     0.000 1.000 0.000
3  bleu     0.000 0.000 1.000
4  rose     1.000 0.750 0.750
5  noir     0.000 0.000 0.000
```

La fonction `mprintf` fait la même chose que `printf` avec une fonctionnalité supplémentaire, la possibilité d'utiliser des *tabulations*, ce qui évite d'avoir à gérer à la main les largeurs de colonne dans un tableau. Une tabulation est codée par la séquence `\t`

```
-->mprintf('%d\t%s\t%.3f\t%.3f\t%.3f\n',(1:5)',couleurs,RGB);
1       rouge   1.000   0.000   0.000
2       vert    0.000   1.000   0.000
3       bleu    0.000   0.000   1.000
4       rose    1.000   0.750   0.750
5       noir    0.000   0.000   0.000
```

C'est évidemment plus confortable pour les gens pressés. Voici le tableau complet :

```
-->mprintf('\n\n%s\t%s\t%s\t%s\t%s\n','no','couleur','R','V','B');...
-->mprintf('%d\t%s\t%.3f\t%.3f\t%.3f\n',(1:5)',couleurs,RGB);


no      couleur R       V       B
1       rouge   1.000   0.000   0.000
2       vert    0.000   1.000   0.000
3       bleu    0.000   0.000   1.000
4       rose    1.000   0.750   0.750
5       noir    0.000   0.000   0.000
```

Ceci dit, il peut y avoir quelques inconvénients à abandonner le contrôle manuel de la largeur de colonne en se reposant sur les tabulations. En voici un exemple.

```
-->mprintf('\n\n%s\t%s\t%s\t%s\t%s\n','no','couleur','R','V','B');...
-->mprintf('%d\t%s\t%.7f\t%.7f\t%.3f\n',(1:5)',couleurs,RGB);
```

```
no      couleur R       V       B
1       rouge   1.0000000       0.0000000       0.000
2       vert    0.0000000       1.0000000       0.000
3       bleu    0.0000000       0.0000000       1.000
4       rose    1.0000000       0.7500000       0.750
5       noir    0.0000000       0.0000000       0.000
```

Il existe aussi des fonctions `fprintf` et `mfprintf` qui agissent exactement comme `printf` et `mprintf`, à ceci près qu'elles écrivent dans un fichier au lieu d'écrire à l'écran, voir la documentation en ligne.

Pour finir, indiquons une variante dans la syntaxe de `printf`, souvent utilisée[5]. Le premier argument est une phrase (donc chaîne de caractères), où chaque occurence d'une variable à afficher est remplacée par sa sous-chaîne de formatage. Les arguments suivants sont les noms des variables dans l'ordre de leur apparition dans la phrase.

```
-->printf('Le prix TTC est de %.2f euros dont %.2f euros de TVA \n...
-->calculée au taux de %.3f',prixht,prixht*taux,taux)
Le prix TTC est de 127.56 euros dont 27.55 euros de TVA
calculée au taux de 0.216
```

5. C'est celle utilisée dans le seul exemple proposé par la doc en ligne de Scilab !