Formation SAS Macros

Martin Chevalier (DMCSI) et Yves Dubois (DESE)

18-19 janvier 2016

Objectifs et organisation pratique

Objectifs :

- acquérir les principes fondamentaux de l'utilisation du macro-langage dans SAS ;
- découvrir certaines utilisations avancées.

Organisation pratique :

- alternance d'éléments théoriques et de cas pratiques ;
- les fichiers de la formation sont stockés sur le lecteur réseau `\\s90sfer1\formation\` dans le sous-dossier `A_Salle_401\Sas_formation_20160118` ;
- des propositions de solutions seront déposées dans ce sous-dossier au fur et à mesure des exercices pratiques.

Présentons-nous !

Objectifs et organisation pratique

De l'intérêt du macro-langage de SAS

Le principal intérêt d'un logiciel de traitement statistique comme SAS est qu'il permet de sauvegarder l'ensemble des opérations effectuées sous la forme d'un **code**.

Cela permet de **reproduire et de contrôler ces opérations** *ex post*, mais au prix d'une certaine « **rigidité** » :

- le moindre changement dans le nom ou l'emplacement des fichiers sources empêche le programme de s'exécuter ;
- certaines opérations répétitives nécessitent de recopier certaines portions de code ;
- l'utilisation est limitée aux opérations prévues par les concepteurs du logiciel (`PROC`).

Le langage macro permet d'introduire une certaine **souplesse** dans le code SAS, en jouant plus ou moins le rôle d'un **langage de programmation**.

De l'intérêt du macro-langage de SAS

Exemple 1 : Centraliser des paramètres en début de code

Certains paramètres sont déterminants pour garantir qu'un code puisse être rejoué par d'autres utilisateurs.

Exemples : emplacement des tables de données, nom et emplacement des fichiers de sortie.

Le langage macro permet de centraliser ces paramètres en début de code par l'intermédiaire de **macro-variables**.

Dans un code bien construit, il suffit de modifier la valeur d'une macro-variable pour adapter un code à un changement d'arborescence.

De l'intérêt du macro-langage de SAS

Exemple 2 : Reproduire un même traitement sur des tables différentes

Il est fréquent que des données soient stockées dans plusieurs tables organisées par année ou par région (RP, DADS, etc.).

Le langage macro permet d'appliquer facilement un même traitement à chacune de ces tables en utilisant des **boucles**.

Cela évite d'avoir à recopier et à adapter à la main le code pour chaque table. En cas de modification, celle-ci s'applique automatiquement à toutes les tables.

Plus encore, l'utilisation de **structures conditionnelles** permet d'adapter les traitements aux spécificités de certaines tables (année avec des variables spécifiques, etc.).

De l'intérêt du macro-langage de SAS

Exemple 3 : Ajouter de nouvelles fonctionnalités à SAS

Toutes les méthodes nécessaires à la production ou à l'exploitation de données statistiques ne sont pas disponibles nativement dans le logiciel.

Exemple : SAS ne dispose d'aucune procédure permettant de mettre en œuvre un calage sur marges.

L'écriture de **macro-programmes** permet néanmoins d'étendre les possibilités du logiciel.

Tout utilisateur peut utiliser des macro-programmes écrits par d'autres et créer ses propres macro-programmes.

Principes fondamentaux du macro-langage

Identification et définition des éléments du macro-langage (1)

Il est facile de repérer les éléments du macro-langage dans un code : ils commencent par les signes `%` ou `&`.

Exemples : `%LET`, `%PUT`, **`%MACRO`**, `&maMacroVariable`, etc.

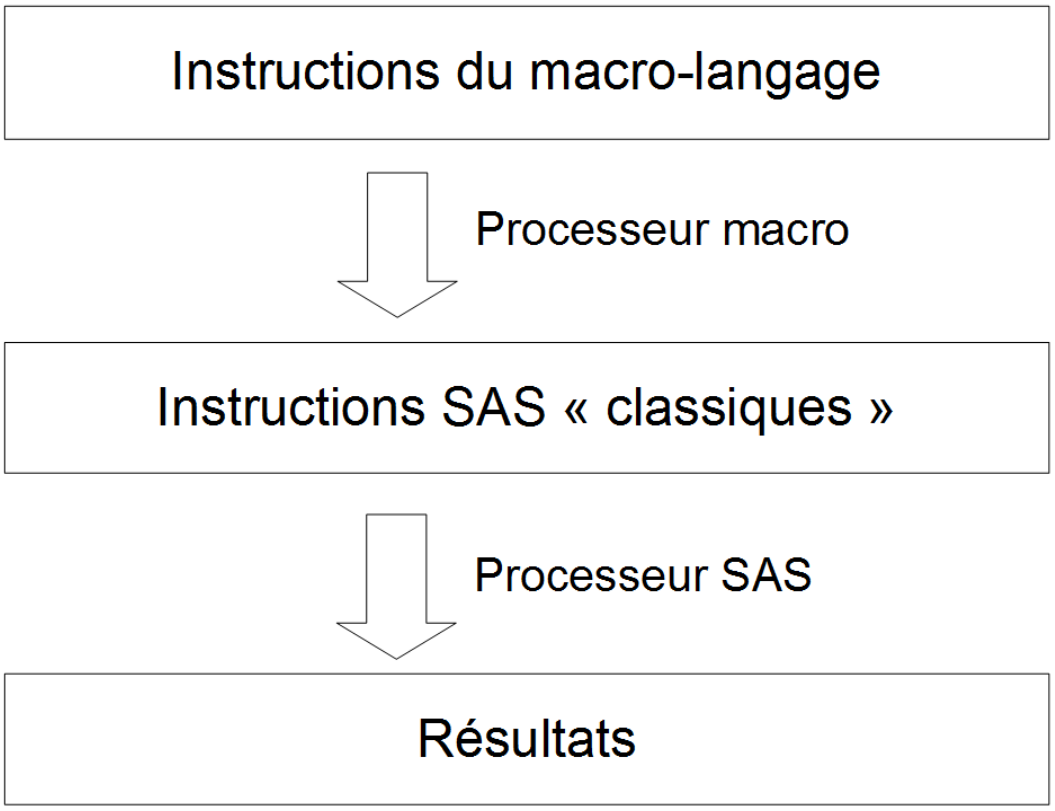
De manière générale, on parle de macro-langage pour désigner un ensemble d'opérations qui s'exécute **avant le langage SAS « classique »** (étape **`DATA`**, **`PROC`**).

Ainsi, quand SAS exécute un code :

1. il repère **d'abord** les éléments du macro-langage et les exécute ;
2. **puis** il exécute les éléments du langage « classique » (étapes **`DATA`**, **`PROC`**).

Principes fondamentaux du macro-langage

Identification et définition des éléments du macro-langage (2)



Principes fondamentaux du macro-langage

Définition et utilisation de macro-variables (1)

L'élément le plus simple du macro-langage est la **macro-variable**.

On définit une macro-variable avec la macro-instruction `%LET` et on affiche sa valeur dans le journal avec la macro-instruction `%PUT` :

```
/*Définition de la macro-variable annee*/
%LET annee = 2016;

/*Affichage de la valeur de annee dans le journal*/
%PUT Année actuelle : &annee.;
```

Principes fondamentaux du macro-langage

Définition et utilisation de macro-variables (2)

Quand elle est appelée, **le nom d'une macro-variable est précédé par `&` et suivi par `.`**. Si le `&` n'est pas présent, elle n'est pas reconnue :

```
138  %LET annee = 2016;
139  %PUT Année actuelle : &annee.;
Année actuelle : 2016
140  %PUT Année actuelle : annee;
Année actuelle : annee
```

Le `.` après le nom de la macro-variable est optionnel : il sert à indiquer la fin du nom de la macro-variable quand celui-ci n'est pas suivi par un espace :

```
5    %LET annee = 2016;
6    %PUT Nom du fichier : eec&annee.1;
Nom du fichier : eec20161
7    %PUT Nom du fichier : eec&annee1;
WARNING: Apparent symbolic reference ANNEE1 not resolved.
Nom du fichier : eec&annee1
```


Principes fondamentaux du macro-langage

Définition et utilisation de macro-variables (3)

Pour qu'une macro-variable soit reconnue au sein d'une chaîne de caractères, il faut utiliser des guillemets double `""` et non des guillemets simples `''` :

```
144  %LET annee = 2016;
145  %PUT Avec des guillemets doubles : "&annee.";
Avec des guillemets doubles : "2016"
146  %PUT Avec des guillemets simples : '&annee.';
Avec des guillemets simples : '&annee.'
```

Cas pratique : Centraliser des paramètres en début de code.

Principes fondamentaux du macro-langage

Définition et utilisation de macro-variables (4)

Il est possible d'afficher toutes les macro-variables définies en utilisant les mots-clés _ALL_ ou _USER_ dans l'instruction %PUT :

```
34   /*Affichage de toutes les macro-variables définies*/
35   %PUT _USER_;
GLOBAL CNIO mvs.cnio.insee.fr
GLOBAL ANNEE 2016
GLOBAL CNIN mvs.cnin.insee.fr
```

À la différence de _USER_, _ALL_ affiche aussi les macro-variables définies automatiquement par le système.

Cas pratique : Afficher les macro-variables système.

Principes fondamentaux du macro-langage

L'exécution d'un code étape par étape (1)

C'est quand langage « classique » et macro-langage interviennent conjointement qu'il importe de bien comprendre comment SAS les exécute.

Exemple :

```
/*Définition de la macro-variable annee*/
%LET annee = 2016;

/*Utilisation de annee dans une PROC FREQ*/
PROC FREQ DATA = matable&annee.;
    TABLES var1*var2;
RUN;
```

Principes fondamentaux du macro-langage

L'exécution d'un code étape par étape (2)

Première étape : exécution du macro-langage

1. affectation de la valeur `2016` à la macro-variable `annee` ;
2. résolution de `&annee.` en `2016` dans la **`PROC FREQ`**.

Seconde étape : exécution du langage « standard »

```
PROC FREQ DATA = matable2016;
    TABLES var1*var2;
RUN;
```

Concrètement, le langage macro écrit le code SAS qui est ensuite exécuté « normalement ».

Principes fondamentaux du macro-langage

Tout ce qui transite par le macro-langage est du texte

D'un point de vue technique, les macro-variables sont des **chaînes de caractères**.

Par défaut, les nombres ne sont donc pas reconnus comme tels :

```
60   %PUT La somme de 2 + 4  est : 2 + 4;
La somme de 2 + 4  est : 2 + 4
```

Pour effectuer des opérations, il faut utiliser la macro-fonction `%SYSEVALF()` :

```
61   %PUT La somme de 2 + 4  est : %SYSEVALF(2 + 4);
La somme de 2 + 4  est : 6
```

Cas pratique : Afficher le temps mis par une série d'opérations.

Principes fondamentaux du macro-langage

Utiliser des macro-fonctions

SAS dispose de fonctions spécifiques pour manipuler les macro-variables : les **macro-fonctions**.

Toute macro-variable étant du texte, ce sont principalement des fonctions de manipulation de chaînes de caractères :

- `%LENGTH()` : déterminer la longueur de la macro-variable ;
- `%UPCASE()` : mettre la macro-variable en majuscules ;
- `%INDEX()` : déterminer la position d'un caractère ;
- `%SUBSTR()` : extraire une sous-chaîne de caractères ;
- `%SCAN()` : extraire un « mot » (pour une liste donnée de séparateurs).

Cas pratique : déterminer la longueur d'une macro-variable et son nombre de « mots ».

Principes fondamentaux du macro-langage

Utiliser des fonctions de l'étape DATA avec %SYSFUNC() (1)

Il est également parfois nécessaire d'utiliser des fonctions « classiques » de SAS pour manipuler des macro-variables.

Exemples :

- utiliser d'autres fonctions de manipulation de chaînes de caractères (par exemple `COMPRESS()`) ;
- arrondir la valeur d'une macro-variable.

La macro-fonction `%SYSFUNC()` permet d'appliquer ces fonctions classiques à une macro-variable.

Principes fondamentaux du macro-langage

Utiliser des fonctions de l'étape DATA avec %SYSFUNC() (2)

Exemple :

```
%LET avecEspace = table_       2016;
%LET sansEspace = %SYSFUNC(COMPRESS(&avecEspace.));

%PUT &avecEspace.;
%PUT &sansEspace.;
```

```
150  %PUT &avecEspace.;
table_       2016
151  %PUT &sansEspace.;
table_2016
```

Cas pratiques : afficher le temps mis par une série d'opérations.

Construire des macro-programmes

Plan de la partie

Construire des macro-programmes

Déclarer un macro-programme (1)

Un macro-programme (ou une « macro ») commence par le mot clé **%MACRO** et se termine par le mot clé **%MEND**.

```
%MACRO mprog();
    ...
    /*Le code SAS que l'on veut exécuter*/
    ...
%MEND mprog;
```

Lorsque l'on exécute les lignes précédentes :

- macro-programme ⇒ macro-compilateur ⇒ stocke le programme dans la `work`.

Construire des macro-programmes

Déclarer un macro-programme (2)

On peut ensuite lancer le macro-programme en saisissant son nom précédé de `%`.

```
%mprog();
```

Le macro-programme stocké dans la `work` va être interprété :

1. macro-programme ⇒ macro-interpréteur ⇒ du texte
2. du texte ⇒ interpréteur SAS ⇒ ...

Construire des macro-programmes

Déclarer un macro-programme (3)

Un macro-programme peut contenir n'importe quel code SAS

```
%MACRO exemple;
    PROC PRINT DATA=class;
        var sex weight;
    RUN;
%MEND exemple;
```

Pour le macro-interpréteur le contenu du programme n'est ici que du texte. Il ne fait rien et renvoie en résultat ce contenu.

Construire des macro-programmes

Déclarer un macro-programme (4)

Ainsi :

```
%exemple();
%exemple();
%exemple();
```

Génère le code SAS suivant :

```
PROC PRINT
    DATA=class;
     VAR sex
         weight;
RUN;
PROC PRINT
    DATA=class;
     VAR sex
         weight;
RUN;
PROC PRINT
    DATA=class;
     VAR sex
         weight;
RUN;
```

Qui va afficher trois fois le contenu de la table `class`.

Construire des macro-programmes

Paramétrer un macro-programme (1)

Il est possible de passer des paramètres à la macro. Ces paramètres sont des macro-variables que l'on peut utiliser dans le corps de la macro. Les paramètres sont déclarés entre `()` après le nom de la macro, et séparés par une virgule.

```
%MACRO copier(tableEntree, tableSortie);
    DATA &tableSortie;
        SET &tableEntree;
    RUN;
%MEND copier;
```

Lorsque l'on exécute la macro :

```
%copier(sashelp.class, matable);
```

celle-ci remplace les macro-variables et produit le code :

```
DATA matable;
    SET help.class;
RUN;
```


Construire des macro-programmes

Paramétrer un macro-programme (2)

Les paramètres passés à un macro-programme peuvent être positionnels ou non-positionnels (avec ou sans valeur par défaut) :

- paramètres positionnels : ils seront obligatoires lorsque l'on utilisera la macro, leurs valeurs seront données dans l'ordre ;
- paramètres non-positionnels : suivis de = dans leur déclaration et d'une valeur par défaut, ils seront facultatifs.

Lorsque la macro a des paramètres positionnels et non-positionels, les paramètres postionnels doivent être déclarés en premier.

Pour les paramètres non-positionnels, la valeur par défaut peut être vide.

Construire des macro-programmes

Paramétrer un macro-programme (3)

Un exemple un peu artificiel :

```
%MACRO mprog2(p1,p2,p3=,p4=,p5=0);
    /*On affiche dans la LOG les paramètres */
    %PUT &p1 &p2 &p3 &p4 &p5;
%MEND mprog2;

%mprog2(p3=3,p4=4);
/*Erreur : p1, p2 obligatoires ! */
%mprog2(1,2,p3=3,p4=4);
/*affiche 1 2 3 4 0*/
%mprog2(1,2);
/*affiche 1 2 0*/
%mprog2(1,2,p4=3,p3=4);
/*affiche 1 2 4 3 0 */
%mprog2(1,2,p5=5,p4=4,p3=3);
/*affiche 1 2 3 4 5 */
```


Construire des macro-programmes

Les boucles itératives (1)

Les macro-programmes peuvent contenir des structures de contrôle qui vont permettre de répéter des parties du code de la macro ou de ne les traiter que pour une condition donnée.

Les répétitions de code sont réalisées avec des boucles `%DO %TO` ou `%DO %WHILE` et l'application de conditions avec des macro-instructions `%IF` et `%ELSE` de manière tout à fait similaire aux structures de contrôle dans les étapes **DATA**.

Construire des macro-programmes

Les boucles itératives (2) : Syntaxe des boucles %DO %TO

```
%DO macrovariable = val_debut %TO val_fin;
    /* code répété */
    /* avec macrovariable incrémentée de 1 à
       chaque itération*/
%END;
```

```
%MACRO exDO();
    %DO i = 1 %TO 5;
        %PUT &i;
    %END;
%MEND;

%exDO();
```

```
%PUT 1;
%PUT 2;
%PUT 3;
%PUT 4;
%PUT 5;
```


Construire des macro-programmes

Les boucles itératives (3) : Syntaxe des boucles %DO %TO

```
%MACRO concatCh(deb,fin);
    DATA chomage;
        SET
%DO i = &deb %TO &fin;
    chomage&i
%END;
        ;
    RUN;
%MEND;
%concatCh(2010,2015);
```

```
DATA chomage;
    SET
        chomage2010
        chomage2011
        chomage2012
        chomage2013
        chomage2014
        chomage2015
    ;
RUN;
```

Attention aux `;` !!!

Construire des macro-programmes

Les boucles itératives (4) : Syntaxe des boucles %DO %WHILE

```
%MACRO test(deb,fin);
    DATA chomage;
        SET
%DO %WHILE (&deb < &fin);
    chomage&deb
    %LET deb = %EVAL(&deb +
       1);
%END;
         ;
    RUN;
%MEND;
```

```
DATA chomage;
    SET
        chomage10
        chomage11
        chomage12
        chomage13
        chomage14
        chomage15
    ;
RUN;
```

Les boucles %DO %WHILE sont moins fréquentes que les boucles %DO %TO, elles ne sont utiles que lorsque la condition d'arrêt de la boucle n'est connue qu'au cours des itérations.

Construire des macro-programmes

Les conditions (1)

```
%MACRO signe(var1);
    %IF(&var1 >= 0) %THEN %DO;
    /* le %PUT ne sera effectué que si var1 est
       positive ou nulle*/
        %PUT Le paramètre est positif;
    %END;
%MEND;
```

Construire des macro-programmes

Les conditions (2)

```
%MACRO signe(var1);
    %IF(&var1 >= 0) %THEN %DO;
        %PUT Le paramètre est positif;
    %END;
    %ELSE %DO;
        /* N'est affichée que si les conditions
           précédentes ne sont pas vérifiées*/
        %PUT Le paramètre est négatif;
    %END;
%MEND;
```


Construire des macro-programmes

Les conditions (3)

Il est possible de chaîner des conditions avec `%ELSE %IF` :

```
%MACRO signe(var1);
    %IF(&var1 > 0) %THEN %DO;
        %PUT Le paramètre est positif;
    %END;
    %ELSE %IF(&var1 = 0) %THEN %DO;
        %PUT Le paramètre est nul;
    %END;
    %ELSE %DO;
        %PUT Le paramètre est négatif;
    %END;
%MEND;
```

Construire des macro-programmes

Les conditions (4)

Il est possible d'imbriquer des structures de contrôle :

```
%MACRO parite();
    %DO I = 1 %TO 30;
        %IF (%SYSFUNC(MODULO(&I,2)) = 1) %THEN %DO;
            %PUT &I est impaire;
        %END;
        %ELSE %DO;
            %PUT &I est paire;
        %END;
    %END;
%MEND;
```


Construire des macro-programmes

Les conditions (5) : Cas des variables non-numériques

```
%MACRO test(var1, var2);
    %IF(&var1 = &var2) %THEN %DO;
        %PUT var1 = var2;
    %END;
    %ELSE %DO;
        %PUT var1 != var2;
    %END;
%MEND;
%test(un,un); /* affiche: var1 = var2 */
%test(un,deux); /* affiche: var1 != var2 */
```

Mais ! ! !

```
%test(+,+); /* affiche une erreur !!! */
%test(=,=); /* affiche: var1 != var2 !!! */
```

Construire des macro-programmes

Les conditions (6) : Cas des variables non-numériques

Solution : protéger les macro-variables pour les comparaisons.

```
%MACRO test(var1, var2);
    %IF(%NRBQUOTE(&var1) = %NRBQUOTE(&var2)) %THEN
       %DO;
        %PUT var1 = var2;
    %END;
    %ELSE %DO;
        %PUT var1 != var2;
    %END;
%MEND;

%test(un,un); /* affiche : var1 = var2 */
%test(un,deux); /* affiche : var1 != var2 */
%test(+,+); /* affiche : var1 = var2 */
%test(=,=); /* affiche : var1 = var2 */
```


Construire des macro-programmes

Déboguer ses macro-programmes

Les options pour déboguer les macros :

- `MPRINT` : affiche dans la LOG le code SAS généré par la macro ;
- `SYMBOLGEN` : affiche dans la LOG chaque résolution de macro-variable ;
- `MLOGIC` : affiche dans la LOG la résolution des boucles `%DO` et des conditionnelles `%IF`.

Les options `NOMPRINT`, `NOSYMBOLGEN` et `NOMLOGIC` annulent respectivement les options précédentes.

```
OPTIONS MPRINT;
%macro1();
OPTIONS NOMPRINT;
```

Construire des macro-programmes

La portée des macro-variables (1)

La notion de portée des macro-variables :

- une macro-variable définie dans une macro est **locale** lorsqu'elle **n'est pas** visible / utilisable / modifiable en dehors de cette macro. Elle sera détruite quand la macro se terminera.
- une macro-variable définie dans une macro est **globale** lorsqu'elle **est** visible / utilisable / modifiable en dehors de cette macro. Elle n'est pas détruite à la fin de la macro.

Pour définir une variable locale on utilise le mot clé `%LOCAL` et `%GLOBAL` pour définir une variable globale.

Construire des macro-programmes

La portée des macro-variables (2)

```
%LET mvar1 = global1;
%LET mvar2 = global2;
%MACRO macro12();
    %GLOBAL mvar1;
    %LOCAL mvar2; /* On crée une variable locale */
    %LET mvar1 = local1; /* On modifie la variable
       globale*/
    %LET mvar2 = local2; /* On modifie la variable
       local*/
    %LET mvar3 = local3; /* La variable est locale
       par défaut */
%MEND;
%macro12;
%PUT &mvar1; /* affiche local1 */
%PUT &mvar2; /* affiche global2 */
%PUT &mvar3; /* Erreur mvar3 n'existe plus  */
```

Construire des macro-programmes

Partager un macro-programme (1)

Il y a plusieurs solutions pour partager un macro-programme :

1. partager le fichier `.sas` avec le code du macro-programme ;
2. partager une bibliothèque dans laquelle le macro-programme est compilé.

La solution la plus simple est d'enregistrer le programme SAS contenant le macro-programme et d'utiliser `%INCLUDE` pour importer ce programme dans les programmes SAS qui utiliseront la macro.

Construire des macro-programmes

Partager un macro-programme (2)

Soumettre le code source a pour effet de compiler et de stocker le macro-programme correspondant dans un catalogue appelé `sasmacr`.

Par défaut, le catalogue `sasmacr` est stocké dans la bibliothèque `work`. Par exemple, la macro compilée générée par l'exécution du code :

```
%MACRO calcul();
    /*...*/
%MEND;
```

est enregistrée dans l'entrée `work.sasmacr.calcul.macro`.

Par défaut, la `work` et donc le compilé de la macro sont perdus en fin de session.

Construire des macro-programmes

Partager un macro-programme (3)

Pour sauvegarder la macro dans une bibliothèque permanente il faut ajouter les options `MSTORED` et `SASMSTORE` :

```
LIBNAME mylib 'Z:\ ';
OPTIONS MSTORED SASMSTORE = mylib;

%MACRO calcul() / STORE;
    /*...*/
%MEND;
```

La macro compilée sera sauvegardée dans le répertoire `Z:\`.

Travailler avec des tableaux et le CALL SYMPUT

Plan de la partie

1. Traitement des `&` par le macro-processeur ;
2. Application : les tableaux de macro-variables ;
3. Le `CALL SYMPUT` dans les étapes **`DATA`** ;
4. Les tables utilitaires :
   - 4.1 `sashelp.vcolumns` et
   - 4.2 `sashelp.vtable`.

Travailler avec des tableaux et le CALL SYMPUT

Traitement des & par le macro-processeur (1)

Fonctionnement du macro-processeur :

- Il remplace `&nom` par le contenu de la macro-variable `nom` ;
- Il remplace `&&` par `&`.
- Le macro-processeur est persévérant ! ! !
  Il relit le texte qui lui est soumis jusqu'à ce qu'il n'y ait plus de `&` qu'il puisse interpréter.

Travailler avec des tableaux et le CALL SYMPUT

Traitement des & par le macro-processeur (2)

```
%LET var1 = un;

PROC PRINT DATA = &var1 ; RUN;

PROC PRINT DATA = un ; RUN;
```

Le macro-processeur passe **PROC PRINT** DATA = un; **RUN;** au processeur SAS.

Travailler avec des tableaux et le CALL SYMPUT

Traitement des & par le macro-processeur (3)

```
%LET var1 = un;
%LET var2 = var1;
%LET i = 2;

PROC PRINT DATA = &&var&i ; RUN;

PROC PRINT DATA = &var2 ; RUN;

PROC PRINT DATA = var1 ; RUN;
```

Le macro-processeur passe `PROC PRINT DATA = var1 ; RUN;` au processeur SAS.

Travailler avec des tableaux et le CALL SYMPUT

Traitement des & par le macro-processeur (4)

```
%LET var1 = un;
%LET var2 = var1;
%LET i = 2;

PROC PRINT DATA = &&&var2 ; RUN;

PROC PRINT DATA = &var1 ; RUN;

PROC PRINT DATA = un ; RUN;
```

Le macro-processeur passe `PROC PRINT DATA = un ; RUN;` au processeur SAS.

Travailler avec des tableaux et le CALL SYMPUT

Application : les tableaux de macro-variables

Si l'on crée un tableau de macro-variables :

```
%LET region1 = dads11;
%LET region2 = dads21;
%LET region3 = dads41;
%LET region4 = dads51;
%LET region5 = dads71;
%LET region_nb = 5;
```

On peut par exemple boucler sur le tableau créé :

```
%MACRO concatReg();
    DATA dadsFR;
        SET
            %DO I = 1 %TO &region_nb;
                &&region&I (KEEP= sexe salaire)
            %END;
        ;
    RUN;
%MEND;
```


Travailler avec des tableaux et le CALL SYMPUT

Créer des macro-variables dans l'étape **DATA** : le CALL SYMPUT (1)

L'instruction CALL SYMPUT permet de créer automatiquement des macro-variables à partir d'une étape **DATA**.

```
DATA _NULL_;
    SET sashelp.class;
    CALL SYMPUT("NbElmnt",_N_);
RUN;
%PUT Nombre lignes de sashelp.class : &NbElemnt;
```

Travailler avec des tableaux et le CALL SYMPUT

Créer des macro-variables dans l'étape **DATA** : le `CALL SYMPUT` (2)

```
%MACRO listMod(TABLE, var);
    PROC SORT DATA = &TABLE OUT = temp(KEEP =
        &var) NODUPKEY;
         BY &var;
    RUN;

    DATA _NULL_;
         SET temp;
         CALL SYMPUTX(CAT("mod",_N_),&var,"L");
         CALL SYMPUTX("NbElmnt",_N_,"L");
    RUN;

    %PUT Modalités de la variable &var;
    %DO I = 1 %TO &NbElmnt;
         %PUT Modalité &I : &&mod&I;
    %END;

%MEND;
```


Travailler avec des tableaux et le CALL SYMPUT

Les tables utilitaires : `sashelp.vtable`

TABLE: Contenu de la table `sashelp.vtable`

Nom de la variable	Descriptif
crdate	Date de création
libname	Bibliothèque
memlabel	Libellé de la table
memname	Nom de la table
modate	Date de modification
nobs	Nombre d'observations
nvar	Nombre de variables
typemem	Type de table (DATA, etc.)

Travailler avec des tableaux et le CALL SYMPUT

Les tables utilitaires : `sashelp.vcolumns`

TABLE: Contenu de la table `sashelp.vcolumns`

Nom de la variable	Descriptif
libname	Bibliothèque
memname	Nom de la table
memtype	Type de table (DATA ou VIEW)
name	Nom de la variable
type	Type de variable (num ou char)
length	Longueur de la variable
varnum	Numéro de variable
label	Libellé de la variable
format	Format de la variable

Travailler avec des listes et la PROC SQL

De l'intérêt de la PROC SQL pour le macro-langage

Comme le `CALL SYMPUT`, la **`PROC SQL`** de SAS permet de créer automatiquement des macro-variables.

Ceci est particulièrement utile dans deux contextes :

- quand la macro-variable doit stocker le résultat d'une fonction d'agrégation ;
  Exemple : nombre d'observations répondant à une certaine condition.
- quand la macro-variable doit stocker une liste de valeurs.
  Exemple : liste des modalités distinctes d'une variable.

Ces deux opérations sont beaucoup plus faciles à réaliser avec une **`PROC SQL`** qu'avec un `CALL SYMPUT`.

Travailler avec des listes et la PROC SQL

Introduction à la PROC SQL (1) : Environnement **PROC SQL**

La **PROC SQL** permet d'utiliser SQL (*Structured Query Langage*) dans SAS :

```
/*Des instructions SAS*/
PROC SQL;
    /*Des instructions SQL*/
QUIT;
/*Des instructions SAS*/
```

Exemple : Table `ventes`

	pays	annee	volume
1	DE	2013	138
2	DE	2014	137
3	DE	2015	139
4	FR	2013	130
5	FR	2014	132
6	FR	2015	131

Travailler avec des listes et la PROC SQL

Introduction à la PROC SQL (2) : Syntaxe de base

Les mots-clés `SELECT` et `FROM` permettent d'afficher dans la fenêtre de résultats les variables d'une base de données :

```
PROC SQL;
    SELECT pays, annee, volume FROM ventes;
QUIT;
```

Note : les noms de variables sont séparés par des « , ».

Il est également possible de sélectionner toutes les variables d'une table avec le caractère spécial `*` :

```
PROC SQL;
    SELECT * FROM ventes ORDER BY annee, pays;
QUIT;
```

Note : `ORDER BY` permet de trier le résultat en sortie.

Travailler avec des listes et la PROC SQL

Introduction à la PROC SQL (3) : Manipulation de données

Le mot-clé `WHERE` permet de restreindre le traitement selon une condition logique :

```
PROC SQL;
    SELECT annee, volume FROM ventes WHERE pays =
        'FR';
QUIT;
```

Pour créer une nouvelle base de données à partir de l'instruction précédente, on ajoute `CREATE TABLE AS` :

```
PROC SQL;
    CREATE TABLE fr AS
        SELECT annee, volume FROM ventes
        WHERE pays = 'FR'
    ;
QUIT;
```

Note : il n'y a qu'un seul « ; » par requête SQL.

Travailler avec des listes et la PROC SQL

Introduction à la PROC SQL (4) : Fonctions d'agrégation

Il est très facile en SQL d'utiliser des fonctions d'agrégation :

```
PROC SQL;
    SELECT MEAN(volume), SUM(volume) FROM ventes;
QUIT;
```

Les fonctions `MEAN()`, `SUM()`, `MIN()`, `MAX()` sont notamment disponibles.

La fonction `COUNT()` compte le nombre de lignes et la fonction `COUNT(DISTINCT mavar)` le nombre de modalités distinctes pour la variable `mavar` :

```
PROC SQL;
    SELECT COUNT(*), COUNT(DISTINCT pays) FROM
        ventes;
QUIT;
```

Travailler avec des listes et la PROC SQL

Introduction à la PROC SQL (5) : Groupements et fusions

Le mot-clé `GROUP BY` indique qu'un traitement doit être effectué selon les modalités d'une ou plusieurs variables :

```
PROC SQL;
    SELECT SUM(volume)
    FROM ventes GROUP BY pays
    ;
QUIT;
```

SQL est également particulièrement utile pour effectuer des fusions complexes entre bases de données multiples :

```
PROC SQL;
    SELECT base1.id1, base2.var1, base3.var2
    FROM base1
        LEFT JOIN base2 ON base1.id1 = base2.id1
        LEFT JOIN base3 ON base2.id2 = base3.id2
    ;
QUIT;
```


Travailler avec des listes et la PROC SQL

PROC SQL et macro-langage (1) : Mot-clé `INTO`

Pour envoyer le résultat d'une instruction SQL dans une macro-variable, il suffit d'utiliser le mot-clé `INTO` :

```
PROC SQL NOPRINT;
    SELECT COUNT(*) INTO :nbLigne FROM ventes;
    SELECT SUM(pays = 'DE') INTO :nbDE FROM ventes;
    SELECT COUNT(DISTINCT pays) INTO :nbPays FROM
       ventes;
QUIT;
%PUT &nbLigne; /*6*/
%PUT &nbDE; /*3*/
%PUT &nbPays; /*2*/
```

Note : le signe `:` précède le nom des macro-variables après `INTO`.

Cas pratique : Compter les observations d'une table.

Travailler avec des listes et la PROC SQL

PROC SQL et macro-langage (2) : Mot-clé SEPARATED BY

Pour stocker plusieurs valeurs dans une macro-variable, il suffit d'ajouter le mot-clé SEPARATED BY avec un séparateur :

```
PROC SQL NOPRINT;
    SELECT DISTINCT pays
    INTO :listePays SEPARATED BY ' '
    FROM ventes
    ;
QUIT;
%PUT &listePays; /*DE FR*/
```

Note : quand l'instruction SQL renvoie plusieurs valeurs et que le mot-clé SEPARATED BY n'est pas utilisé, par défaut seule la première valeur est affectée à la macro-variable.

Travailler avec des listes et la PROC SQL

PROC SQL et macro-langage (3) : Affectations multiples

Plusieurs macro-variables peuvent être créées par la même instruction SQL :

```
PROC SQL NOPRINT;
    SELECT DISTINCT pays, COUNT(DISTINCT pays)
    INTO :listePays SEPARATED BY ' ', :nbPays
    FROM ventes
    ;
QUIT;
%PUT &listePays; /*DE FR*/
%PUT &nombrePays; /*2*/
```

Les noms des macro-variables après le mot-clé INTO sont dans ce cas séparés par des virgules.

Travailler avec des listes et la PROC SQL

Travailler avec des listes (1)

Comme les tableaux de macro-variable, les macro-variables de listes sont particulièrement indiquées pour appliquer le même traitement à plusieurs tables ou variables.

La macro-fonction `%SCAN(texte,pos)` est dans ce cadre essentielle :

```
%LET listeReg = 11 21 22 91;
%PUT %SCAN(&listeReg,2); /*21*/
%PUT %SCAN(&listeReg,3); /*22*/
```

Pour un ensemble donné de délimiteurs, `%SCAN(texte,pos)` extrait le « mot » en position `pos` de `texte`.

Travailler avec des listes et la PROC SQL

Travailler avec des listes (2)

Par défaut, `%SCAN()` considère les caractères suivants comme des délimiteurs : `blank ! $ % & ( )+ - */ . , ; < ^`.

Le troisième argument optionnel de `%SCAN()` permet d'indiquer explicitement le ou les délimiteurs à prendre en compte :

```
%LET liste = 1er élément. $$$ 2nd élément.;
%PUT %SCAN(&liste,1,$$$); /*1er élément.*/
%PUT %SCAN(&liste,2,$$$); /*2nd élément.*/
```

Note : les éléments extraits par la fonction `%SCAN()` peuvent donc contenir des espaces (modalités d'une variable caractère) ou des points (table stockée dans une bibliothèque).

Travailler avec des listes et la PROC SQL

Travailler avec des listes (3)

Couplée à une boucle `%DO`, la fonction `%SCAN()` permet ainsi de parcourir les éléments d'une liste :

```
%MACRO boucle(listeReg=11 21 22 91,nbReg=4);
    %DO numReg = 1 %TO &nbReg;
        %LET idReg = %SCAN(&listeReg,&numReg);
        %PUT &idReg;
    %END;
%MEND boucle;
%boucle;
```

Une liste, définie manuellement ou à l'aide d'une instruction SQL, permet ainsi d'appliquer facilement le même traitement à un ensemble de variables ou de tables.

Cas pratiques : Renommer en bloc les variables d'une table, travailler avec une liste de tables, construire un programme de dichotomisation automatique.

Travailler avec des listes et la PROC SQL

Éléments avancés (1) : Les tables `dictionary`

Les vues `sashelp.vtable` et `sashelp.vcolumns` mentionnées précédemment sont inaccessibles depuis la **`PROC SQL`**.

Néanmoins, les tables `dictionary.tables` et `dictionary.columns` jouent exactement le même rôle :

- `dictionary.tables` comporte des informations sur toutes les tables accessibles à l'utilisateur ;
- `dictionary.columns` comporte des informations sur toutes les variables des tables accessibles à l'utilisateur.

Cela permet de détecter automatiquement les tables présentes dans une bibliothèque ou les variables présentes dans une table.

Travailler avec des listes et la PROC SQL

Éléments avancés (1) : Les tables `dictionary`

Exemple :

```
PROC SQL NOPRINT;
    SELECT memname
    INTO :listeTable SEPARATED BY ' '
    FROM dictionary.tables
    WHERE libname = 'WORK'
    ;
    SELECT name
    INTO :listeVariable SEPARATED BY ' '
    FROM dictionary.columns
    WHERE libname = 'WORK' AND memname = 'VENTES'
    ;
QUIT;
```

Cas pratiques : Renommer en bloc les variables d'une table, travailler avec une liste de tables.

Travailler avec des listes et la PROC SQL

Éléments avancés (2) : Génération de code

Il est également possible d'utiliser la **PROC SQL** pour générer directement du code SAS dans une macro-variable :

```
PROC SQL NOPRINT;
    SELECT DISTINCT
        CAT(pays," = (pays = '",pays,"');")
    INTO :codeDicho SEPARATED BY ' '
    FROM ventes
    ;
QUIT;
DATA ventes;
    SET ventes;
    &codeDicho
    /*DE = (pays = 'DE'); FR = (pays = 'FR');*/
RUN;
```