

# Bon code, bad code



```
\\ \| ///  
\\ - - //  
( @ @ )  
ooo ( _ ) ooo
```

**Mais, qu'est-ce que du "bon code" SAS®?**

```
Oooo  
oooo ( )  
( ) ) /  
\ ( ( /  
\ ) -  
-
```



Alain Voyer  
utilisateur SAS® curieux  
et explorateur de code

Club des utilisateurs SAS de Québec, 3 mai 2017

# Algorithme principal

Introduction;

Objectifs\_d'apprentissage;

On\_jase\_là;

Concept\_de\_«distance\_cognitive»;

do until(épuisement\_des\_stocks);

    exemple;

    code\_original;

    code\_proposé;

    conclusion\_partielle;

end;

Quelques\_sujets;

Conclusion?;

# Introduction

Programmer: art ou science ?

Coder est un acte de création personnel

Le code possède une forte empreinte du concepteur

- Dépend des connaissances (le savoir)
  - Apprentissage formel (cours, groupes d'utilisateurs)
  - Auto-apprentissage (documentation, exemples des autres)
- Dépend de l'expérience (le savoir-faire)
  - Fréquence, « intensité » d'utilisation
  - Profil de l'utilisateur (informaticien, statisticien, etc.)
  - Relation avec d'autres langages (c, java, basic, assembler, etc.)
- De l'environnement de travail (le contexte)
  - Niveau de maturité de l'organisation (gestion des T.I.)
  - Travail en équipe, partage de code, documentation



# Introduction

Programmer: art ou science ?

Coder est un acte de création personnel

- Possède une forte empreinte du concepteur
  - Style et contexte de programmation
    - Plateformes utilisées, sources de données
    - Nomenclature.
    - Majuscules/minuscules, indentation, commentaires, etc.
    - Structuration, modularité, cohérence

# Objectifs d'apprentissage

- Comment améliorer sa programmation SAS en bâtissant une discipline personnelle de fonctionnement.
- Qu'est-ce que la « distance cognitive » dans la programmation SAS et comment la réduire pour améliorer sa performance dans la création de traitements SAS.
- Comment développer des réflexes de conception de programmes pour obtenir des résultats plus robustes et plus résilients.

# On jase là...

## Du bon code pour qui pour quoi?

- Pour l'humain

- Ergonomie mentale (lisibilité, compréhension, concision, etc)



- Pour « l'ordinateur »

- Efficience (volume de données, temps d'exécution, consommation de ressources, etc.)



Le code est conçu par un humain mais exécuté par l'ordinateur

(le code peut aussi être généré)

Plus on « taponne » un programme plus on risque de le contaminer (i.e. introduire des bugs)



# On jase là...

## Processus de création

Comment abordez-vous la création de vos traitements SAS?

Comment choisissez-vous le nom de vos tables SAS, de vos variables, de vos fichiers, de vos macros, etc.

Utilisez-vous les noms originaux de vos sources de données (i.e. bases de données)?

Suivez-vous des normes?

Vous fiez-vous à votre instinct?

Prenez-vous exemple sur votre voisin-e?

Réutilisez-vous vos traitements?

Consultez-vous la documentation?



# Distance cognitive

intro



La **cognition** est le terme scientifique qui sert à désigner l'ensemble des processus mentaux qui se rapportent à la fonction de connaissance tels que la mémoire, le langage, le raisonnement, l'apprentissage, l'intelligence, la résolution de problème, la prise de décision, la perception ou l'attention.

Source: <https://fr.wikipedia.org/wiki/Cognition>

# Distance cognitive

tentative de definition personnelle

Niveau d'effort fait par le cerveau pour faire le lien entre 2 ou plusieurs éléments.

Grande distance =====> Effort élevé

Petite distance ===> Effort moindre

La distance cognitive peut être modulée par aussi par le dénivelé.

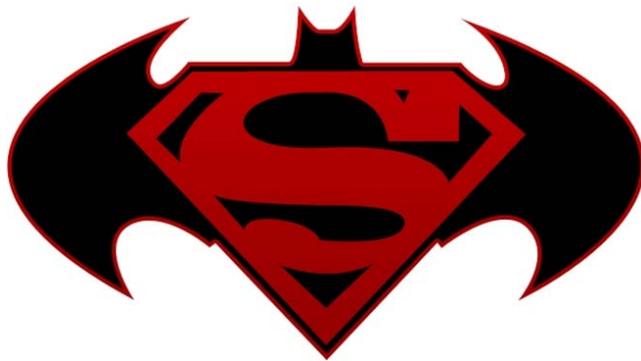
Comme à vélo, le niveau de difficulté d'un parcours est lié à la combinaison entre la distance et le dénivelé.

# Distance cognitive



Exemples fantaisistes – rupture de sens

```
data Batman;  
  infile Superman;  
  etc...;  
run;
```



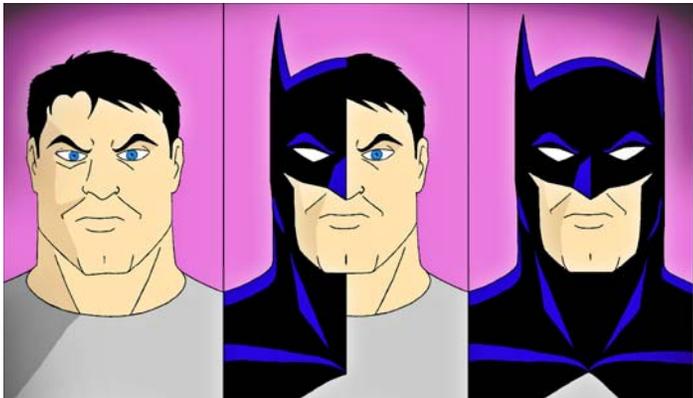
```
data Superman;  
  infile Batman;  
  etc...;  
run;
```



# Distance cognitive

Exemples fantaisistes – plus cohérents non?

```
data Batman;  
  infile BrWayne...;  
  etc;  
run;
```



```
data Superman;  
  infile ClKent...;  
  etc;  
run;
```



# Distance cognitive



Exemples réels – noms peu significatifs:

```
data temp;  
  infile lecture;  
  etc;  
run;
```

```
data donnees;  
  infile fichier;  
  etc;  
run;
```

# Distance cognitive

aspect "dénivelé"



Ouf!

Pouf pouf!

Arrgh!



```
data xyz;  
proc sort;  
proc print;
```



```
proc tabulate;  
proc transpose;
```



```
proc ds2;  
proc fcmp;
```



# Distance cognitive

Garder la distance cognitive au minimum

- Nommer les choses pour mieux comprendre
  - pour soi et pour les autres
- Utiliser des noms significatifs (mais encore!)
  - la sémantique, la syntaxe
- Garder les noms originaux de la documentation des systèmes de l'organisation
  - et des sources de données en général si possible
- Préserver la cohérence dans les noms utilisés
  - dans le traitements différents

# Distance cognitive

## nomenclature



### Exemple:

```
data tEtudiant;
  drop _: ;
  infile Etudiant;
  _ExamFin=scan(1st, -1);
  if _ExamFin='R' then;
  etc;
run;
proc sort data=tEtudiant
          out=tEtudiantT;
  by Code..;
run;
```

Création de la table SAS des étudiants. Le préfixe t indique que c'est une table SAS.

À partir du Fichier des étudiants

On se débarrasse des variables intermédiaires avec le DROP

Les variables intermédiaires sont facilement repérables

Trier la table des étudiants

Le suffixe T indique que la table est triée

# Traitement de dates



code original

```
DATJOUR = DATE();
```

```
* ON CHERCHE LE DERNIER JOUR DU MOIS PRECEDENT ;
```

```
IF MONTH(DATJOUR) = 3 THEN
```

```
    IF MOD(YEAR(DATJOUR),4)=0 THEN DERJR=29;
```

```
        ELSE DERJR=28;
```

```
ELSE
```

```
    IF MONTH(DATJOUR) IN (5,7,10,12) THEN DERJR=30;
```

```
        ELSE DERJR=31;
```

# Traitement de dates



## code proposé

\* ON CHERCHE LE DERNIER JOUR DU MOIS PRECEDENT ;

\* En version 9 on peut utiliser;

```
DERJR = day(intnx("month",DATE(),-1,"end"));
```

\* Avec la version 5 on peut utiliser;

```
DERJR=day(intnx("days",  
              mdy(month(date()),1,year(date())),-1));
```

Format de la fonction INTNX en version 5:

```
INTNX(interval, from, number)
```

Format simplifié de la fonction INTNX en version 9:

```
INTNX(interval, start-from, increment <, 'alignment'>)
```

# Traitement de dates



## conclusion partielle

### Le traitement des dates est une force de SAS

- Algorithme personnel
  - Fastidieux (if then else) et verbeux (lignes de code)
  - Sujet à erreur: algorithme incomplet (pour février année bissextile)
- Fonctions de dates en SAS
  - Simple et concis: un énoncé avec fonctions
  - Fiable : algorithme fiable et complet (de 1582 à l'an 19 900!)
- Donc: consultons la documentation et utilisons les fonctions!
  - Pas seulement pour les dates
- Transposition d'un langage autre vers SAS
  - Bénéfice ou piège?

# Traitement de chaînes de caractères

## exemple 1

Extrait d'un journal de système (syslog):

```
-----1-----2-----3-----4-----5-----6-----7-----8-----9-----0-----1-----2-----
N 0000000 GEST      2017111 00:00:26.76 STC28403 00000290 DRKUX07I TM00,C,,,TGSEP1IA,UNX44540,$SOPERA,1704202300,2351,1704210000
N 0000000 GEST      2017111 00:00:26.76 STC28403 00000290 DRKUX07I TM00,C,,,TGSEP1IA,UNX44540,$SOPERA,1704202300,2351,1704210000
S
N 0000000 GEST      2017111 00:00:27.27 STC28403 00000290 DRKUX07I TM00,C,,,QDMCE15P,JOB37013,$SYSINX,1704202359,2359,1704210000
S
N 0000000 GEST      2017111 00:00:25.38 STC37040 00000291 IEF404I SLGGEST - ENDED - TIME=00.00.25
N 0004000 GEST      2017111 00:00:25.38 STC37040 00000290 -SLGGEST ENDED. NAME- TOTAL CPU TIME= .00
S
N 4000000 GEST      2017111 00:00:25.38 STC37040 00000291 $HASP395 SLGGEST ENDED - RC=0000
N 0000000 GEST      2017111 00:00:25.39 00000290 IEA989I SLIP TRAP ID=X33E MATCHED. JOBNAME=*UNAVAIL, ASID=0085.
N 0000000 GEST      2017111 00:00:27.99 STC32630 00000290 OPS3092H OI MVSSCHK0
N 0000000 GEST      2017111 00:00:28.03 STC32630 00000291 OPS1181H OPSOSF OPSS (*LOCAL*) MVS N/A MVSSCHK0 D SMF
NC0000000 GEST      2017111 00:00:28.03 GEXTCS02 00000290 D SMF
-----1-----2-----3-----4-----5-----6-----7-----8-----9-----0-----1-----2-----
```

Partie à structure fixe: colonnes 1 à 58:

```
N 0000000 GEST      2017111 00:00:26.76 STC28403 00000290
```

Partie variable: colonnes 59 à 132:

```
DRKUX07I TM00,C,,,TGSEP1IA,UNX44540,$SOPERA,1704202300,...
```

# Traitement de chaînes de caractères

## Ex. 1 - code original

Extrait d'un journal de système (syslog):

Enregistrement no. 1: var1A,var1B,var1C,var1D

Enregistrement no. 2: var2A,var2B,,var2D

```
ValeurA=scan(LigneLog, 1);
```

```
ValeurB=scan(LigneLog, 2);
```

```
ValeurC=scan(LigneLog, 3);
```

```
ValeurD=scan(LigneLog, 4);
```



Enreg.	ValeurA	ValeurB	ValeurC	ValeurD	LigneLog
1	var1A	var1B	var1C	var1D	var1A,var1B,var1C,var1D
2	var2A	var2B	var2D		var2A,var2B,,var2D

Syntaxe SAS V5: SCAN(argument1, n <,delimiters >)

# Traitement de chaînes de caractères

## Ex. 1 - code proposé

Extrait d'un journal de système (un syslog):

Portion d'enregistrement no. 1: var1A,var1B,var1C,var1D

Portion d'enregistrement no. 2: var2A,var2B,,var2D

```
ValeurA=scan(LigneLog, 1, ", ", "m");
```

```
ValeurB=scan(LigneLog, 2, ", ", "m");
```

```
ValeurC=scan(LigneLog, 3, ", ", "m");
```

```
ValeurD=scan(LigneLog, 4, ", ", "m");
```



Enreg.	ValeurA	ValeurB	ValeurC	ValeurD	LigneLog
1	var1A	var1B	var1C	var1D	var1A,var1B,var1C,var1D
2	var2A	var2B		var2D	var2A,var2B,,var2D

Syntaxe SAS V9: SCAN(string,count <,character-list <,modifier>>)

# Traitement de chaînes de caractères

## Ex 1. - conclusion partielle

Plusieurs fonctions SAS possèdent maintenant un argument supplémentaire:

le « modifier » qui en altère le fonctionnement.

On le retrouve entre autres dans ces fonctions:

- COMPRESS(source <,characters> <,modifier(s)>)
- FIND(string, substring <,start-position> <,modifier(s)>)
- COUNT(string, substring <,modifier(s)>)
- COMPARE(string-1, string-2 <,modifier(s)>)



Pour SCAN le modificateur “m” ou “M”:

« specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the string argument, refer to words that have a length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the string argument are ignored. »

Encore une fois, cette possibilité peut nous aider à simplifier le code.

au lieu de faire une boucle DO avec des INDEX et des SUBSTR

# Traitement de chaînes de caractères

## Exemple 2

But: enlever les zéros après le point

1.02.002.002.001.001

2.04.005.06.107.002

3.04.050.020.009

4.03.030.014

Pour obtenir:

1.2.2.2.1.1

2.4.5.6.107.2

3.4.50.20.9

4.3.30.14

# Traitement de chaînes de caractères

## Ex. 2 - code original

```
IF SUBSTR(CODE,17,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,17) || SUBSTR(CODE,19,02);  
IF SUBSTR(CODE,17,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,17) || SUBSTR(CODE,19,02);  
IF SUBSTR(CODE,16,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,16) || SUBSTR(CODE,18,03);  
IF SUBSTR(CODE,16,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,16) || SUBSTR(CODE,18,03);  
IF SUBSTR(CODE,15,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,15) || SUBSTR(CODE,17,04);  
IF SUBSTR(CODE,15,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,15) || SUBSTR(CODE,17,04);  
IF SUBSTR(CODE,14,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,14) || SUBSTR(CODE,16,05);  
IF SUBSTR(CODE,14,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,14) || SUBSTR(CODE,16,05);  
IF SUBSTR(CODE,13,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,13) || SUBSTR(CODE,15,06);  
IF SUBSTR(CODE,13,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,13) || SUBSTR(CODE,15,06);  
IF SUBSTR(CODE,12,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,12) || SUBSTR(CODE,14,07);  
IF SUBSTR(CODE,12,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,12) || SUBSTR(CODE,14,07);  
IF SUBSTR(CODE,11,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,11) || SUBSTR(CODE,13,08);  
IF SUBSTR(CODE,11,02)=' .0'  
  THEN CODE = SUBSTR(CODE,1,11) || SUBSTR(CODE,13,08);
```

etc, etc.

60 lignes de code au total

# Traitement de chaînes de caractères

## Ex. 2 - code proposé

```
data EnleveZero;  
  input CodeLu $char20.;  
  CodeTran=CodeLu;  
  CodeTran=transtrn(CodeTran, ".00", ".");  
  CodeTran=transtrn(CodeTran, ".0", ".");  
  ....;  
run;
```

### Liste ces codes originaux et transformés

Obs.	CodeLu	CodeTran
1	1.02.002.002.001.001	1.2.2.2.1.1
2	2.04.005.06.107.002	2.4.5.6.107.2
3	3.04.050.020.009	3.4.50.20.9
4	4.03.030.014	4.3.30.14

# Traitement de chaînes de caractères

## Ex. 2 – conclusion partielle

Le code répétitif est un indice qu'on peut utiliser une autre stratégie pour arriver à nos fins.

C'est ce qu'on peut appeler de la propagation verticale de code.

Plus il y a de lignes de codes plus il y a de possibilités d'erreurs

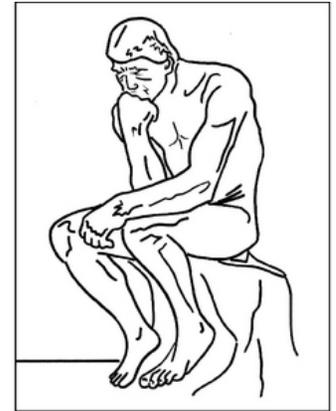
D'autres stratégies sont possibles pour arriver au même résultat:

- Boucles
- Fonctions
- Langage macro
- Formats standards ou fabriqués

Ça demande cependant de faire des recherches dans la documentation ou de consulter des collègues.

# Nomenclature

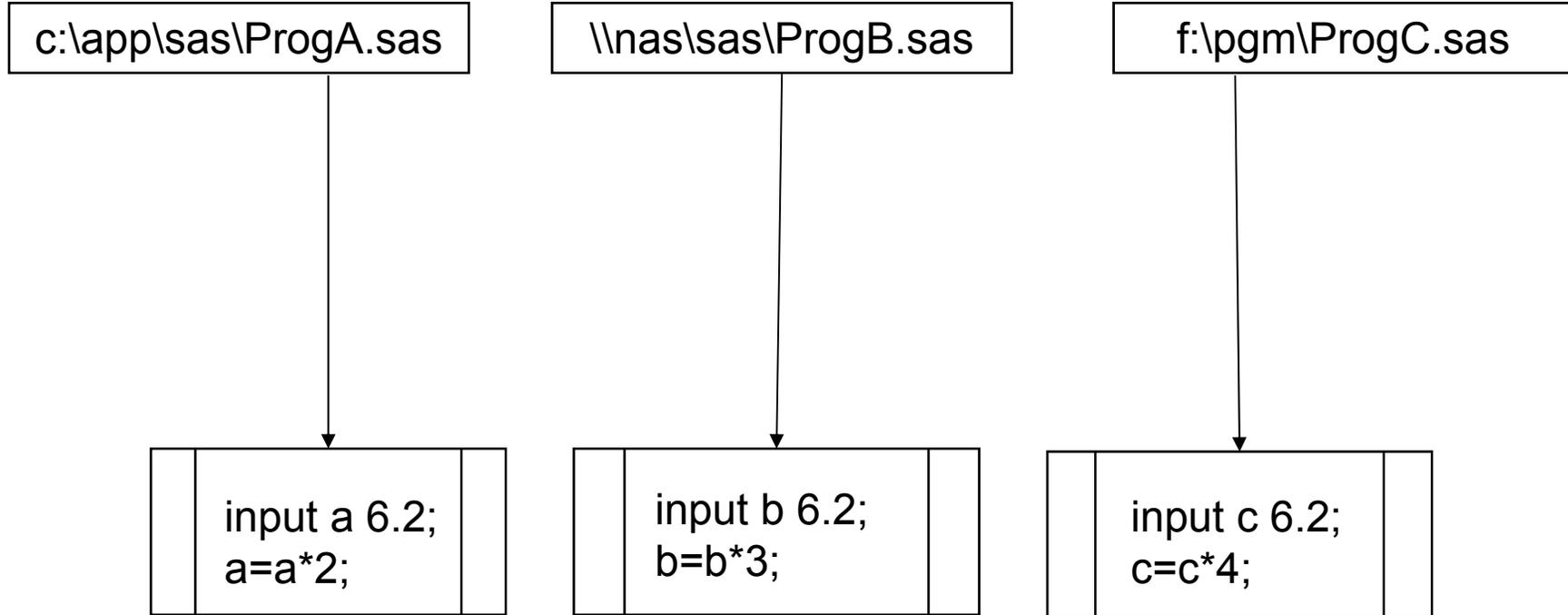
donner du sens aux mots



- Noms des fichiers et bases de données
- Noms des tables SAS
- Noms des variables (data, macro)
- Noms des routines, macros
- L'effort mental de nommer les choses (la sémantique) permet de mieux comprendre le traitement.
- La cohérence dans la nomenclature favorise une meilleure « fluidité » mentale (ergonomie mentale, cohérence sémantique et syntaxique).
- Garder autant que possible les noms originaux dans les fichiers et bases de données

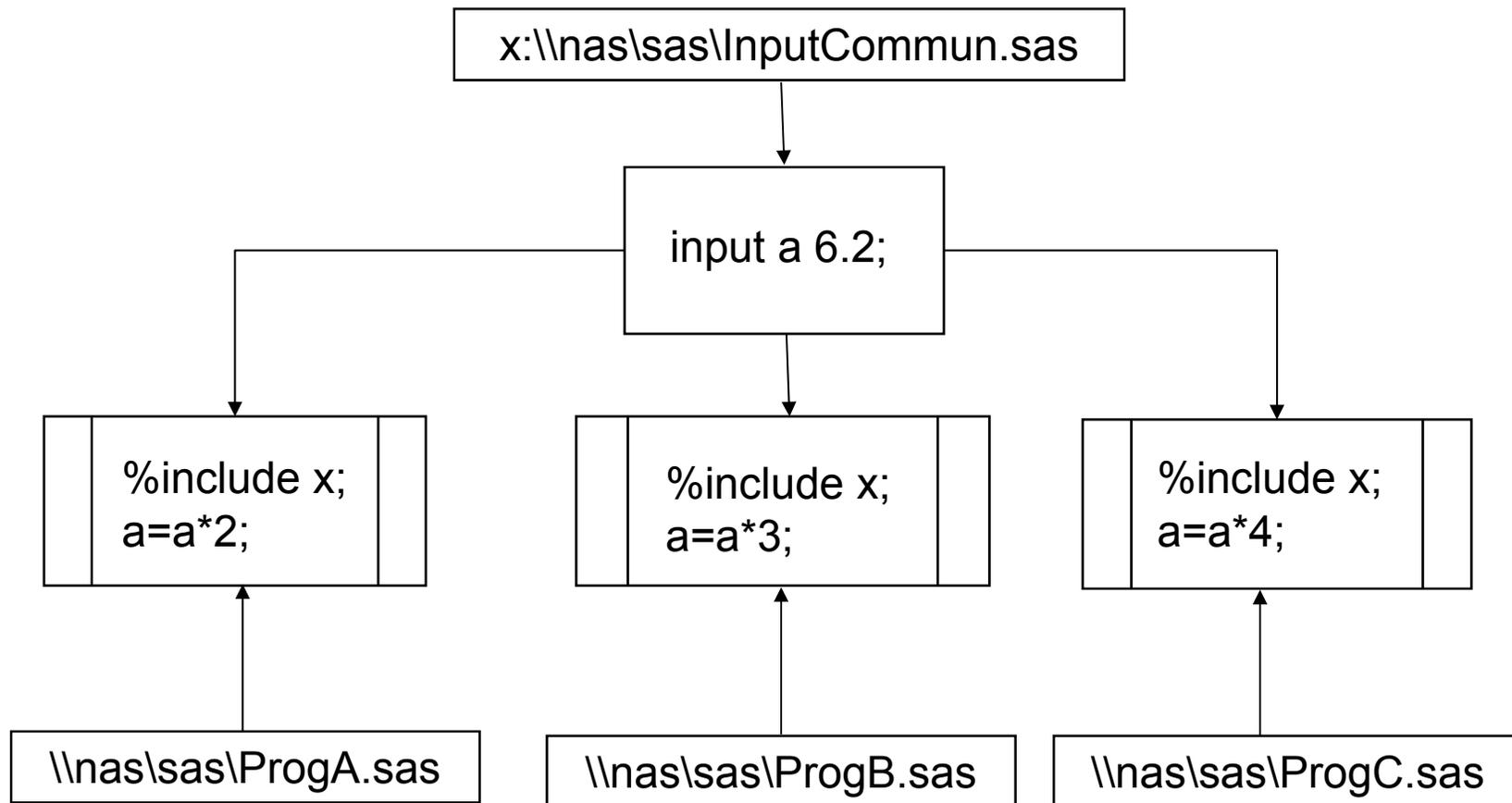
# Structure de lecture

sans partage de code



# Structure de lecture

## avec partage de code



# Codage “en dur” définition

- Consiste à imbriquer des dépendances fortes aux données à l’intérieur du traitement SAS
- Pratique courante mais pouvant devenir problématique
  - Nécessite de retoucher plus fréquemment le code
    - Augmente les coûts d’entretien
    - Augmente la possibilité d’erreurs
  - Peut ne pas fournir de bons résultats
    - Données pas à jour dans le traitement
  - Ne permet pas une cohérence de résultats dans une organisation
    - Valeurs de données différentes d’un programme à l’autre



# Codage “en dur” symptômes

- Détection du codage en dur
  - Le ratio du nombre de modifications sur le nombre d'exécutions
    - $1/1 = 100\%$ ,  $1/10 = 10\%$ ,  $1/100 = 1\%$
    - Objectif: garder le ratio au plus bas
  - Autre indices
    - Présence de données dans le programme
    - Exemple: une série de IF en cascade ou une série de SELECT
  - Pistes de solutions:
    - PROC FORMAT avec CNTLIN
    - La paramétrisation
    - Les tables du dictionnaire SAS



# Codage “en dur”

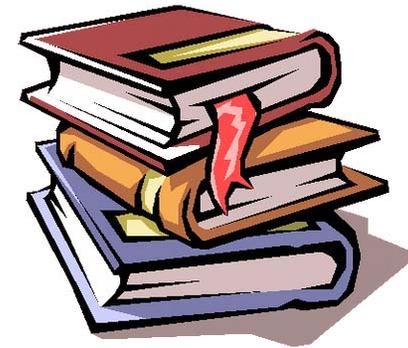
## exemple avec le dictionnaire

Code avant:

```
title1 "Fichier source: /folders/myfolders/DataHC642215s.txt";
```

Code après:

```
proc sql noprint;
  select xpath length=90
  into :NomFsource
  from dictionary.extfiles
  where fileref="FSOURCE"
  ;
quit;
title1 "Fichier source: &NomFsource";
```



Résultat:

**Fichier source: /folders/myfolders/DataHC642215s.txt**

# Codage “en dur”

## exemple avec %sysfunc

Code avant:

```
title3 "Production de TGL0475P le 3 mai 2016";
```

Code après:

```
title3 height=9pt  
      "Production de &sysjobid le "  
      "%sysfunc(left(%sysfunc(date()),eurdfwkx.))" à "  
      "%sysfunc(left(%sysfunc(time()),time5.2))"  
      ;
```

Résultat:

**Production de TGL0475P le 3 mai 2016 à 10:20**

# Laisser des traces...

## pour mieux comprendre

- Utiliser des « putlog » dans vos traitements:

```
putlog "INFO: (pgm)      " ValeurA=;  
putlog "NOTE: (pgm)     " ValeurB=;  
putlog "WARNING: (pgm)  " ValeurC=;  
putlog "ERROR: (pgm)   " ValeurD=;
```



- Une forme d'auto-documentation à l'exécution
  - Laisse des traces utiles lors du développement et en cas d'erreur
  - Coloration comme pour les messages SAS habituels:

```
INFO: (pgm)      ValeurA=var1A  
NOTE: (pgm)     ValeurB=var1B  
WARNING: (pgm)  ValeurC=var1C  
ERROR: (pgm)   ValeurD=var1D
```

# Conclusion?

- Développer son intuition, sa méthode de travail, sa cohérence
  - Éviter la multiplication de traitements similaires
    - Propagation horizontale
    - Favoriser le partage de code, la paramétrisation
  - Éviter le code similaire qui se répète
    - Propagation verticale
    - Favoriser les boucles, les formats, les fonctions
    - Utiliser le langage macro
  - Miser sur les forces de SAS
    - Traitement des dates
    - Transformation de données
  - Utiliser les sources de données d'origine quand c'est possible
    - Les données transformées peuvent réserver des surprises

# Sagesse ancestrale

Thus spake the master programmer:

***« Though a program be but  
three lines long, someday it  
will have to be maintained. »***

© The Tao of Programming, Geoffrey James





*Questions ?  
Commentaires ?  
Suggestions ?*

***Merci!***

Contact:



```
1 data _null_;  
2   courriel=lowcase(cat(reverse("Alain"),reverse("Voyer"),"@gmail.com"));  
3   putlog courriel=;  
4 run;
```

courriel=nialareyov@gmail.com

#### Marques de commerce

SAS et tous les autres noms de produits et services de SAS Institute Inc. sont des marques de commerce de SAS Institute Inc. aux États-Unis et dans les autres pays. ® indique l'inscription au États-Unis. Les autres noms de produits ou marques sont des marques déposées ou des marques de commerce de leurs compagnies respectives.