

# Webinar@Lunchtime

## Eine kleine Auswahl von Stolperfallen im Data Step



# Demo

Quizfragen, Antworten und Erklärungen zu ausgewählten Stolperfallen

# Inhaltsübersicht

- Hintergrundinformation Data Step: Kompilierungs- und Ausführungsphase (Nr. 1, 2)
- Reihenfolge Optionen (z. B. SET-Anweisung) (Nr.3)
- OUTPUT-Anweisung (Nr.4)
- Einlesen bei SET und subsetting-IF (Nr. 5)
- Standardlängen bei Funktionen (Nr.6)
- Automatische Typkonvertierung (Nr.7)
- Besonderheiten bei retain und Variablenlisten (Nr.8)
- Makroverarbeitung: Besonderheit bei CALL SYMPUTX (Nr.9), CALL EXECUTE (Nr.10) und %IF-Anweisungen innerhalb des DATA Steps
- Dateien verbinden: MERGE mit many-to-many Verknüpfung (Nr.12)
- Iterative Schleifen (Nr.13)

# 1. Wie sieht die neue Datei TOGETHER aus?

```
Data Together;  
Merge table1  
      table2;  
By id;  
Run;
```

```
data table1;  
  id='1';  
  name='Huber';  
run;
```

Table 1

ID	Name
1	Huber

+

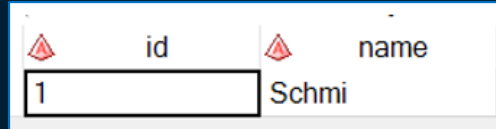
```
data table2;  
  name='Schmitt';  
  id='1';  
run;
```

Table 2

Name	ID
Schmitt	1

# Wie sieht die neue Datei TOGETHER aus?

Antwort: ID=1, Name = Schmi    Länge von Name=5B    Nur 1 Zeile!

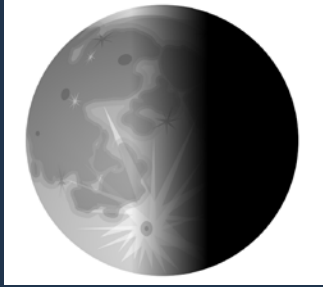


id	name
1	Schmi

## Data Step Backstage:

1. Kompilierungsphase: **METADATEN**-Verarbeitung  
Die **zuerst** eingelesene Variable bestimmt die Länge (name=Huber, 5B)
2. Ausführungsphase: **DATEN**-Verarbeitung  
Gleicher Spaltenname in beiden Dateien und nicht in der BY-Anweisung:  
Vorheriger Wert wird überschrieben, **letzter** Wert wird ausgegeben!

# Data Step:Backstage



2 Phasen im Data Step:

1. **Kompilierungsphase**
2. Ausführungsphase

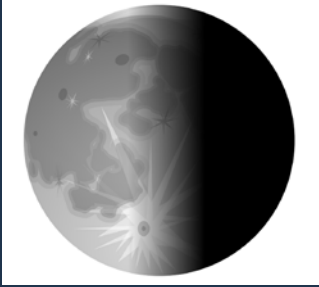
## 1. Kompilierungsphase

- Nur **Metadaten**-Verarbeitung
- SAS geht 1-mal komplett durch die Syntax bis **run**;
- SAS schreibt die Metadaten in den SAS Zwischenspeicher (PDV), die SAS **zuerst** findet (keine spätere Korrektur möglich z. B. Länge, Typ)

Beispiel für PDV

ID	NAME	LAND	...
N 8	\$ 5	\$ 2	

## Data Step: Backstage



2 Phasen im Data Step:

1. Kompilierungsphase
2. **Ausführungsphase**

## 2. Ausführungsphase

Alle Variablen erhalten einen Anfangswert (=Initialisierungswert): **MISSING**

- **Daten an sich** werden verarbeitet
- Daten können überschrieben werden – falls ja: **letzter** Wert wird in die Ausgabedatei geschrieben
- SAS geht n-mal durch die Einlesedatei – so oft wie Anzahl Zeilen (automatische Schleife)
- **Automatische** Ausgabe bei run;

Beispiel für PDV:

ID	NAME	LAND	...
N 8	\$ 5	\$ 2	...
1	Huber	DE	...

## Weitere Anwendungsfälle

**Mehrere WHERE-Anweisungen:** SAS verarbeitet nur **eine** WHERE-Anweisung und nimmt die **letzte**

(links: age > 15 wird verarbeitet)

**LENGTH-Anweisung:** Wenn sie nicht rechtzeitig angegeben wird, dann nimmt SAS die Länge, die SAS **zuerst** findet (land=Großbri), die Length-Anweisung ist zu spät, Italien bestimmt die Länge (=7B)

In der **Makrovariablen** (&mvname) ist nur 1 Wert: das Alter aus der **letzten** Zeile der Einlesedatei (die Einlesedatei sashelp.class hat 19 Zeilen, vorherige Werte werden überschrieben)

```
Data test;  
    set sashelp.class;  
    where name="Alfred";  
    Where age >15;  
Run;
```

```
data neu;  
    land='Italien';  
    land='Großbritannien';  
    length land $ 14;  
run;
```

```
data makrovar_erstellen;  
    set sashelp.class;  
    call symputx('mvname',age);  
run;
```



## Data Step

Sashelp.class (Auszug)

Name	Age	Height
Alfred	14	69
Alice	13	56.5
Barbara	13	65.3
Carol	14	62.8
Henry	14	63.5
James	12	57.3
Jane	12	59.8

## 2. Wie viele Zeilen sind in der neuen Datei?

```
data test;  
  if _n_ = 1 then set sashelp.class(obs=7);  
run;
```

# Wie viele Zeilen sind in der neuen Datei?

2 identische Zeilen

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alfred	M	14	69	112.5

1. Iteration: `if _n_=1:` **wahr**, 1. Zeile wird gelesen ➔ **output**

2. Iteration: Keine Reinitialisierung der Variablen, vorherige Werte bleiben im PDV erhalten  
`if _n_=1:` **falsch**,  
bei `run;` der Inhalt der 1. Zeile ist noch immer im PDV und wird nochmals ausgegeben ➔ **output 2. Zeile**

## 3. Verarbeitungsreihenfolge

Im Data Step kann man relativ frei in der Reihenfolge programmieren – eine logische Abhängigkeit muss beachtet werden.

Es gibt manchmal eine interne Verarbeitungsreihenfolge, die nicht sichtbar ist, aber zu Überraschungen führen kann.

## SET-Anweisung und Optionen

### 3. Wie müssen die SET- und Keep-Anweisung korrekt ergänzt bzw. angegeben werden?

Set A (rename=(name=Firstname) **Keep=???** );  
**Keep ???** ;

- A) set A (... keep=name) ;      keep name;
- B) set A (...keep=Firstname);      keep name;
- C) set A (...keep=name);      keep firstname;
- D) set A (...keep=Firstname);      keep firstname;

# Wie müssen die SET- und Keep-Anweisung korrekt ergänzt bzw. angegeben werden?

Antwort: c

Set A (rename=(name=Firstname) **Keep=name** );

Keep **firstname** ;

1. SAS liest zuerst die Variable ein und kann sie erst dann umbenennen  
(Optionen-Reihenfolge geschrieben ist egal)



Dann ist der neue Name im Zwischenspeicher PDV



2. Keep Anweisung: wird später verarbeitet,  
der neue umbenannte Name muss angegeben werden!

## 4. Output-Anweisung

Die **Output**-Anweisung ermöglicht

- Zeilenvervielfachung
- Ausgabe in mehrere Dateien
- **Explizite** Anweisung:  
Wenn OUTPUT mind. 1-mal in der Syntax, dann macht SAS keine automatische Ausgabe mehr bei **run**;

5a) Wie sieht die neue Datei SECTION aus für Zeile 1 und 2 (Variablen: a und B)?

Output-Anweisung

```
Data section;  
  Do a=1 to 6;  
    Output;  
    B=a;  
  End;  
Run;
```

## Wie sieht die neue Datei SECTION aus?

```
Data section;  
  Do a=1 to 6;  
    Output;  
    B=a;  
  End;  
Run;
```

a	B
1	.
2	1
3	2
4	3
5	4
6	5

Die OUTPUT-Anweisung **innerhalb** der Schleife **vor** B=a:  
Verhindert die Ausgabe von B in der gleichen Iteration.

B= erhält immer den Wert aus dem PDV der vorherigen Iteration



## Ausführungsphase

Verarbeitung im PDV

Am Anfang der  
**Ausführungsphase:**  
alle Variablen haben  
den Startwert:  
MISSING  
(Standardeinstellung)

## Mehr Details (optional): 2 Schleifendurchläufe

1. Schleifendurchlauf **BIS** Output

A	B
1	.

1. Schleifendurchlauf **NACH** Output

A	B
1	1

2. Schleifendurchlauf **BIS** Output

A	B
2	1

# SET-Anweisung Subsetting-If

a	B
1	.
2	1
3	2
4	3
5	4
6	5

```
Data final;  
set section;  
if a >=3 ;  
Set section(keep=b);  
Run;
```

5b) Die vorherige Datei (SECTION) wird weiter verarbeitet.

Wie sieht die ganz neue Datei FINAL aus?

A

A	B
4	3
5	4
6	5

B

A	B
4	.
5	1
6	2

C

A	B
4	.
5	5
6	6

D

A	B
4	.
5	.
6	.

a	B
1	.
2	1
3	2
4	3
5	4
6	5

```
Data final;
  set section;
  if a >=3 ;
  Set section(keep=b);
Run;
```

Wie sieht die ganz neue Datei FINAL aus?  
Antwort: B

FINAL	
a	B
4	.
5	1
6	2

Data FINAL:

1. Set-Anweisung: Wegen if – erst ab 4 ausgeben

Die 2. SET-Anweisung liest ab der ERSTEN Zeile ein (keep=B)

SAS Funktionen:  
Standardlängen beachten

Hier: Substr-Funktion  
Werte verbinden: ||

## 6. Wie ist der genaue Wert von Package zum Schluss?

```
data surprise;  
  package="SAS";  
  part1=substr(package,1,1);  
  part2=substr(package,2,1);  
  part3=substr(package,3,1);  
  package=part3 || part2 || part1;  
run;
```

# Wie ist der genaue Wert von Package zum Schluss?

Antwort: PACKAGE= S \_\_ (S BLANK BLANK)

## Metadaten-Information:

Zuerst gefundene - wird nicht mehr geändert!

Neue Variablen erstellen mit der **SUBSTR-Funktion:**

Länge ist wie Input-Variable (hier: 3B)

## Am Ende:

Package hat noch immer die Länge 3B!

Metadaten werden nicht überschrieben

Package="SAS" - Länge 3B

Beispiel für alle PARTn-Variablen:

Part1 = substr(package,1,1) ;

Part1 = S Blank Blank (Länge 3B)

Package = Part3 || part2 || part1;

Package = S Blank Blank

# Standardlängen einiger SAS Funktionen bei Erstellung neuer Variablen

Ausgewählte Funktionen

Name	Standardlänge	Erklärung
Substr	Wie Input-Variable	Zeichen extrahieren
Scan	Wie Input-Variable	Wörter extrahieren
Tranwrd	200B	Wörter/Zeichen ersetzen
Symget	200B	Makrovariablenwert in einer SAS Variablen speichern
Cats, catx, cat, catt	200B	Werte/Variablen verbinden

## Achtung: Standardlänge bei Funktionen

CAT-Funktionen: Standardlänge 200B

Beispiel:

Var1=600B      Var2=800B

Neu=CATS(var1,var2)

Das Ergebnis wird bei der Standardlänge **200B** abgeschnitten.



LENGTH-Anweisung benutzen!

SAS Funktionen

Weiteres Beispiel

## 7. Welche Ergebniswerte haben die Variablen A und B?

A) A= .      B=1.234

B) A=1234   B=1.234

C) A=.      B=123.4

D) A=1234  
B=123.400000000

```
Data test;  
    Original=1234;  
    A=Input(Original,4.);  
    B=Input(Original,13.3);  
Run;
```



# Welche Ergebniswerte haben die Variablen A und B?

Antwort: A, A=. B=1.234

Die Input-Funktion braucht alphanumerische Werte für die Input-Variable, aber hier: ORIGINAL= numerisch



**Automatische Typkonvertierung zu alphanum.**

(BEST12. Informat - rechts ausgerichtet!)

\_\_\_\_\_1234 (8 führende Leerzeichen)

Input(original,4.) = . (Position 1-4: Blanks)

Input(original,13.3)= 1.234 (SAS nimmt an, dass die letzten 3 Zeichen Nachkommastellen sind)

Explizite Funktionen zur  
Typkonvertierung:  
Keine Meldung im Log

Put(Variable,Format.)

Input(Variable,Informat.)

## Automatische Typkonvertierung

SAS kann in bestimmten Fällen eine **automatische** Typkonvertierung vornehmen:

- Erzeugt eine zusätzliche Meldung im Log
- **Numerisch zu alphanumerisch**  
**BEST12.-Format (rechtsausgerichtet)**
- **Alphanumerisch zu numerisch:**  
**w.d-Einleseformat (z. B. 1234.56, -12E4, keine Kommas!)**  
nur, wenn die Werte keine „störenden alphanumerischen“ Zeichen enthalten  
(Ziffern 0-9, Minuszeichen, Dezimalpunkt,E)

**Achtung: Automatische Typkonvertierung kann zu unerwünschten Ergebnissen führen!**

## Retain-Anweisung Variablenlisten

- A) X=0      L=3  
   Y=AB     L=2
- B) X=.      L=8  
   y=AB     L=2
- C) X=.      L=3  
   Y=        L=2

## 8. Welche Längen und Werte haben X und Y? (Welches ist die richtige Antwort?)

```
Data test2;  
  Retain _numeric_ 0  _character_ 'ABC';  
  Length X 3  Y $ 2;  
Run;
```

## Welche Längen und Werte haben X und Y?

Antwort: C / X= L=3 , Y= L=2

Length:

Zur Kompilierungszeit

Retain:

Zur Kompilierungs- UND Ausführungszeit

`_numeric_`

`_character_`

Variablen, die **momentan** im  
Zwischenspeicher PDV existieren!

**Nicht:** Variablen, die **später** durch  
Programmierung erstellt werden

# Welche Längen und Werte haben die Variablen X und Y?

Antwort: C / X=. L=3 , Y= L=2

- **Keine numerischen /alphanum.** Variablen **existieren**, wenn die RETAIN-Anweisung ausgeführt wird
- Daher: X und sind durch die RETAIN-Anweisung nicht betroffen  
(Kein Initialisierungswert, kein Behalten der Werte (Retain))

## 9. Wo werden die Makrovariablen test1 und test2 erstellt?

- A) Beide: global
- B) Beide: local
- C) Test1: global  
Test2: local
- D) Test1: local  
Test2: global
- E) Nur Test1 wird erstellt  
(global)

```
%macro challenge;  
Data _null_ ;  
  Call symputx('test1' , 'global or local');  
Run;  
  
Proc sql;  
  Select * from sashelp.class;  
quit;  
  
Data _null_ ;  
  Call symputx('test2' , 'where am I');  
Run;  
%mend challenge;
```

# Wo werden die Makrovariablen test1 und test2 erstellt?

Antwort: C    test\_1 global ,    test\_2 local (weil vorher SQL Code steht)

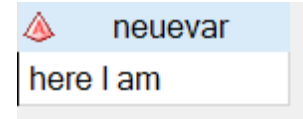
Achtung interne Verarbeitungsreihenfolge bei CALL SYMPUTX:

1. Call Symputx sucht **ZUERST** nach einer bestehenden Variable mit dem jeweiligen Namen – egal ob lokal oder global
  2. Wenn Call Symputx die Variable nicht findet, erstellt Call Symputx die Makrovariable in der „CLOSEST EXISTING“ Symbol Tabelle.
- **CALL Symputx erstellt keine Symbol Tabellen – sondern nutzt nur existierende!!**
- 
- Proc SQL erstellt eine lokale Symbol Tabelle





# 10. Welche Syntax erzeugt NEUEVAR bei 1-maliger Ausführung des Data Steps?



Data test;      ??

A) **call execute** ('data \_null\_ ; call symputx ('mvar', "here I am"); run; ');  
neuevar=symget('mvar');

B) **rc=dosubl** (cats("data \_null\_ ;call symputx ('mvar','here I am'); run;"));  
neuevar= symget('mvar');

C) **call symputx** ('mvar', 'here I am');  
neuevar= "&mvar";

Run;

# Welche Syntax erzeugt NEUEVAR bei 1-maliger Ausführung des Data Steps?

Antwort: B

A) Call execute

B) `rc= dosubl`

C) `call symputx('mvar',  
neuevar= "&mvar";`

- **Call execute:** wird **NACH** dem DATA Step ausgeführt, der das CALL EXECUTE beinhaltet (zu spät!)
- **Dosubl-Funktion:** wird **SOFORT** ausgeführt!!! 😊  
(rc=freigewählter Variablenname)
- **&mvar:** Der Makroprozessor versucht &mvar aufzulösen, **BEVOR** der Data Step mit call symputx läuft (zu früh)

# 11. Welche %IF-Anweisung INNERHALB eines DATA Steps erzeugt die Log Meldung: Hello Monday ?

```
Data test;
```

```
A) %If &sysday = Monday %then %put Hello Monday;
```

```
B) %If &sysday = "Monday" %then put "Hello Monday" ;
```

```
C) %If &sysday = "Monday" %then %do; %put Hello Monday ; %end;
```

```
D) %If &sysday = Monday %then %do; Put "Hello Monday" ; %end;
```

```
Run;
```

# Welche %IF-Anweisung INNERHALB eines DATA Steps erzeugt die Log Meldung: Hello Monday ?

Antwort: D

- Makro %IF Anweisungen außerhalb eines Makros (neu ab SAS 9.4M5), mit Einschränkung:
  - Keine verschachtelten If-Anweisungen
  - %DO ... %END muss sein!
- Auch OPEN CODE ist möglich (außerhalb eines jeden Schrittes)

```
%if &sysday=Monday %then %do;  
  %put Hello Monday;  
%end;
```

```
Data test;  
  %if &sysday= Monday %then %do;  
    Put „Hello Monday“ ;  
  %end;  
Run;
```

# 11. Optional: %IF-Anweisungen in OPEN CODE (Beispiel)

Neu ab SAS9.4M5

- Warum kann es sinnvoll sein, %IF-Anweisungen in OPEN CODE zu benutzen?
- Beispiel: Das Vorhandensein einer Datei prüfen, bevor sie verarbeitet werden soll.

```
%IF %Sysfunc(Exist(work.DataSet)) %THEN %DO;  
  proc print data=work.DataSet;  
  run;  
%END;
```

## Data Step: MERGE Many-to-many Verknüpfung

```
Data NEW;  
Merge A B ;  
By COUNTRY;  
Run;
```

12. Wie viele Zeilen hat die neue Datei, die durch die MERGE Syntax erzeugt wird?

TABLE: A			TABLE: B	
First	Gender	Country	Country	Phone
Togar	M	AU	AU	+61 (2) 5555-1500
Kylie	F	AU	AU	+61 (2) 5555-1600
Stacey	F	US	AU	+61 (2) 5555-1700
Gloria	F	US	US	+1 (305) 555-1500
James	M	US	US	+1 (305) 555-1600

# Wie viele Zeilen hat die neue Datei, die durch die MERGE Syntax erzeugt wird?

Antwort: 6 Zeilen

Many-to-many Merge

A			B	
First	Gender	Country	Country	Phone
Togar	M	AU	AU	+61 (2) 5555-1500
Kylie	F	AU	AU	+61 (2) 5555-1600
Stacey	F	US	AU	+61 (2) 5555-1700
Gloria	F	US	US	+1 (305) 555-1500
James	M	US	US	+1 (305) 555-1600

NEW

First	Gender	Country	Phone
Togar	M	AU	+61 (2) 5555-1500
Kylie	F	AU	+61 (2) 5555-1600
Kylie	F	AU	+61 (2) 5555-1700
Stacey	F	US	+1 (305) 555-1500
Gloria	F	US	+1 (305) 555-1600
James	M	US	+1 (305) 555-1600

## Erklärung

Was passiert ab der Stelle, an der es keine weiteren Übereinstimmungen in der anderen Datei gibt?

SAS verknüpft nur die eine **LETZTE** Zeile der einen Datei mit allen weiteren Übereinstimmungen aus der anderen Datei 😞

# Wie viele Zeilen hat die neue Datei, die durch die MERGE Syntax erzeugt wird?

Optionales Beispiel: Proc SQL – 12 Zeilen 😊

First	Gender	Country		Country	Phone
Togar	M	AU	↔	AU	+61 (2) 5555-1500
Kylie	F	AU	↔	AU	+61 (2) 5555-1600
Stacey	F	US	↔	AU	+61 (2) 5555-1700
Gloria	F	US	↔	US	+1 (305) 555-1500
James	M	US	↔	US	+1 (305) 555-1600



First	Gender	Country	Phone
Togar	M	AU	+61 (2) 5555-1500
Togar	M	AU	+61 (2) 5555-1600
Togar	M	AU	+61 (2) 5555-1700
Kylie	F	AU	+61 (2) 5555-1500
Kylie	F	AU	+61 (2) 5555-1600
Kylie	F	AU	+61 (2) 5555-1700
Stacey	F	US	+1 (305) 555-1500
Stacey	F	US	+1 (305) 555-1600
Gloria	F	US	+1 (305) 555-1500
Gloria	F	US	+1 (305) 555-1600
James	M	US	+1 (305) 555-1500
James	M	US	+1 (305) 555-1600

```
Proc sql;  
  create table NEW as  
  select *  
  From A, B  
  Where A.country=B.country;  
Quit;
```

Many-to-many Verknüpfungen:

Proc SQL erstellt kleine kartesische Produkte mit den übereinstimmenden Werten.

Sinnvoller 😊



### 13. Was ist der Wert von i zum Schluss?

```
Data Loop_test;  
  Do i=1 to 5;  
    i=i*3;  
  end;  
Run;
```

# Was ist der Wert von i zum Schluss?

Antwort:  $i=13$

```
Data Loop_test;  
  Do i=1 to 5;  
    i=i*3;  
  end;  
Run;
```

## Erklärung- **Iterative Schleifen:**

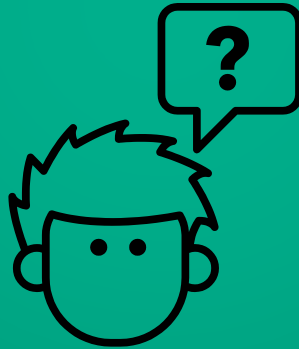
Am Ende einer jeden Schleifeniteration wird die Schleifen-Indexvariable um 1 Increment erhöht (default=1) bei end.

1. Loop:  $i=3$  bei END: +1  $i=4$

2. Loop:  $i=12$  bei END: +1  $i=13$

Wenn  $i=13$ , dann stoppt die Schleife, weil der Wert außerhalb des gültigen Bereichs ist.

Fragen?



# Ready to go: Lernportale zu SAS® 9.4 und SAS® Viya®



Als Einstieg oder Vertiefung in SAS 9.4 Foundation oder SAS Viya, sind unsere beiden Lernportale genau das Richtige für Sie.

Über die Portale erreichen Sie alle E-Learnings zu SAS 9.4 oder SAS Viya sowie eine Auswahl an kostenlosen Tutorial-Videos.

In jedem der Portale sind bis zu 20 Kurse und über 70 Tutorials verfügbar.

# SAS Kurse im März für Kurzentschlossene

## SAS® Programming 1: Essentials

01.-03. März als Live Web Class (Englisch)

22.-24. März als Live Web Class

## SAS® Makrosprache 1: Grundlagen

01.-03. März als Live Web Class

## SAS® Enterprise Guide® 2: Advanced Tasks and Querying

03.-04. März als Live Web Class (Englisch)

## SAS® Programming 2: Data Manipulation Techniques

04.-05. März als Live Web Class (Englisch)

## SAS® Visual Analytics for SAS® 9: Fast Track

08.-11. März als Live Web Class

## SAS® Platform Administration: Fast Track

15.-18. März als Live Web Class

## SAS® Programming 3: Advanced Techniques

15.-19. März als Live Web Class (Englisch)





Nächstes Webinar@Lunchtime:

25. März 2021

[www.sas.de/lunchtime](http://www.sas.de/lunchtime)



Folien zum Download unter [www.sas.de/lunchtime](http://www.sas.de/lunchtime)

Besten Dank für Ihr Feedback 😊

[sas.com](http://sas.com)