
# Webinar@Lunchtime

## Fun & Learn mit SAS Programmier-Quizzes



THE POWER TO KNOW.

# Demo

1. Data Step Syntax und Proc SQL
2. Makro Syntax

--

Schleifen Einstieg

# Wie sieht die Datei aus?

```
data florida;
        do tourists=1 to 3;
                do city='Tampa', 'Miami', 'Orlando';
                    output;
                end;
        end;
   if tourists=3;
run;
```

## Erklärung

Erklärung: Tampa hat 5 B Länge und ist der erste Wert, den SAS findet. Deshalb wird die Länge von CITY auf 5 B gesetzt, Orlando ist aber länger und wird daher abgeschnitten.

Tipp: LENGTH Anweisung verwenden:
length city $ 7;

# Lösung: 9 Zeilen, der Wert von Orlando wird abgeschnitten

| | tourists | city |
|---|---|---|
| 1 | 1 | Tampa |
| 2 | 1 | Miami |
| 3 | 1 | Orlan |
| 4 | 2 | Tampa |
| 5 | 2 | Miami |
| 6 | 2 | Orlan |
| 7 | 3 | Tampa |
| 8 | 3 | Miami |
| 9 | 3 | Orlan |

Schleifen-Hintergrund Wissen

# Was ist der Wert von i zum Schluss?

```
data schleife;
  do i=1 to 5;
   i=i*3;
  end;
run;
```

# Schleifen – Erklärung 1

Bei jedem Schleifendurchlauf am Ende wird die Schleifen-Index Variable um ein Inkrement erhöht.

Default: Inkrement=1

Was sind die Werte der Index-Variablen in den folgenden DO-Anweisungen?

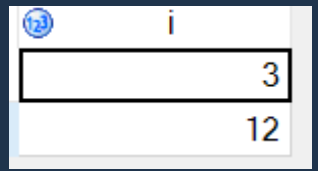
```
do i=1 to 5;
     1 2 3 4 5 6

do j=2 to 8 by 2;
     2 4 6 8 10

do k=10 to 2 by -2;
     10 8 6 4 2 0

do m=3.6 to 3.75 by .05;
     3.6 3.65 3.7 3.75 3.8
```

**Die außerhalb des Bereichs liegenden Werte sind hervorgehoben.**

## Erklärung 2

Hinweis: Wenn eine Output-Anweisung in der Schleife steht, wird die letzte Erhöhung um ein Inkrement nicht mehr in die Datei ausgegeben:

```
Do i=1 to 5;
 i=i*3;
 Output;
End;
```




# Lösung: i=13

1. Schleifendurchlauf **i=1**

   i=i*3; jetzt: i=3 und wird um 1 Inkrement (hier: +1) erhöht also: i=4

   end;

2. Schleifendurchlauf i=4, passt noch zur Bedingung (do i=1 to 5).

   i=i*3 also 4*3=12
   wieder um 1 Inkrement erhöhen: i=13!

Für 13 gibt es zwar keinen Schleifendurchlauf mehr, aber der letzte Wert ist 13.

## Frage 3: Selbststudium

Schleifen

# Was ist der Wert von ***total*** zum Schluss?

```
data test  ;
   retain date '05jan1960'd  start 1;
      do i=start to date;
         start=3;
         date=8;
         total=total + i + 1;
      end;
    total + i;
  run;
```

Frage 4

Arrays

Summen Funktion

# Was ist der Wert von ***total*** zum Schluss?

```
data test ;
   array amounts {20} var1-var20;
      do i=50 to 98 by 5;
         amounts{i/5}= i;
      end;
    total=sum(var18-var20);
run;
```

Arrays:

- Temporär
- Nur jeweils 1 Datentyp möglich
- Nur im Data Step
- Ist KEINE Variable

# Erklärung: Arrays

Definition:

**Arrays Name{Anzahl Element} var1 var2 … var-n ;**

Array amounts{20} var1-var20;

**Hier:** Der eindimensionale Array (ähnlich wie ein Vektor) mit Namen ***amounts*** beinhaltet die Variablen var1-var20.

```
do i=50 to 98 by 5;
amounts{i/5}= i;
```

Hier: Jedem Array-Element wird ein Wert zugewiesen;

Das Array Element (50/5)=10, also var10 bekommt den Wert von i =50; alle vorherigen (var1-var9) haben einen Missing.

Erklärung

# Lösung: Total=Missing

| var18 | var19 | var20 | i | total |
|---|---|---|---|---|
| 90 | 95 | . | 100 | . |

Es ist ein Unterschied, ob man schreibt:

total=sum(var18 – var20) ➡ also (90 - .)

Jede arithmetische Operation mit einem Missing ergibt wieder einen Missing.

Oder ob die Werte in der Klammer mit Komma getrennt sind!

Total=sum(var18,-var20)

Damit die Sum-Funktion den Missing ignoriert, muss das Komma stehen! Dann wäre total = 90!

Frage 5: Tabellen verknüpfen

Data Step: Merge

# Wie sieht die neue Datei ***zusammen*** aus?

table1

| ID | Name |
|---|---|
| 1 | Huber |

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

table2

| Name | ID |
|---|---|
| Schmitt | 1 |

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

```
data zusammen;
    merge table1
          table2;
    by id;
run;
```

Wie viele Zeilen und Spalten hat die Datei?
Welche Werte?

Erklärung

PDV=
Program Data Vector

# Hintergrund: Data Step

## 1. Kompilierungsphase (Metadaten-Verarbeitung)

SAS geht die ganze Syntax EINMAL bis zum ***run;*** durch
Alle Variablen werden jetzt mit Name, TYP, LÄNGE
in den Zwischenspeicher (sog. PDV) geschrieben.
Was zuerst an Länge oder Typ gefunden wird, kann dann nicht mehr geändert werden, z. B. darf deshalb die Length Anweisung nicht zu spät stehen!

## 2. Ausführungsphase (Daten an sich werden verarbeitet)

- SAS gibt jeder Variablen einen Anfangswert (Default=Missing)
- Eine Daten-Zeile nach der anderen durchläuft die Syntax

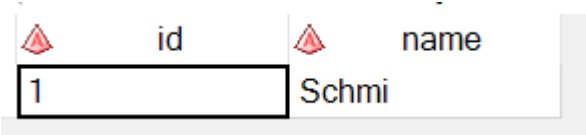
# Lösung – Erster Teil: 1 Zeile, 2 Spalten

Data Step „Backstage“:

1.Kompilierungsphase

2. Ausführungsphase

Variablennamen müssen EINDEUTIG sein, daher kann es keine 2 Name Variablen geben




1. In der **Kompilierungsphase** wird u.a. die Länge der Variablen festgelegt. Es wird die Länge aus der ERSTEN Datei genommen, hier: **Huber** – also 5 Bytes.

2. In der **Ausführungsphase** wird zuerst der Wert Huber eingelesen, dann mit **Schmi** überschrieben (Länge=5Bytes!)

3. SAS nimmt die Metadaten, die ZUERST gefunden werden und überschreibt die Daten an sich mit den Werten, die ZULETZT stehen.

## Data Step: Hintergrund

Kompilierungsphase: Variablen werden mit Name, Typ und Länge im Zwischenspeicher PDV angelegt. **SAS nimmt die zuerst gefundenen Informationen** und überschreibt nicht die Länge oder den Typ, falls später z. B. längere Werte stehen!

# Lösung – Zweiter Teil: Achtung beim Merge

Wenn in den zu verknüpfenden Dateien Variablen vorkommen mit dem gleichen Namen und man will sie nicht in der BY-Anweisung zum Verknüpfen benutzen, dann müssen sie umbenannt werden.

```
data zusammen1;
 merge table1 (rename=(name=name1))
       table2;
       by id;
       run;
```

Andernfalls werden sie mit der Länge von der **zuerst** gefundenen Variablen angelegt (hier: aus table1, Länge 5B) und dann in der Ausführungsphase überschreibt der letzte Wert (hier: Schmi) alle vorherigen.

Frage 6: Dateien verknüpfen

Many-to-many Verknüpfung

# Data Step vs Proc SQL: Wie viele Zeilen erzeugt der Data Step und wie viele Proc SQL?

**EmpsAUUS**

| First | Gender | Country |
|---|---|---|
| Togar | M | AU |
| Kylie | F | AU |
| Stacey | F | US |
| Gloria | F | US |
| James | M | US |

**PhoneO**

| Country | Phone |
|---|---|
| AU | +61(2)5555-1500 |
| AU | +61(2)5555-1600 |
| AU | +61(2)5555-1700 |
| US | +1(305)555-1500 |
| US | +1(305)555-1600 |

```
data EmpsOfc;
   merge EmpsAUUS
         PhoneO;
   by Country;
run;
```

```
proc sql;
   create table EmpsOfc as
   select First, Gender, PhoneO.Country,
          Phone
   from EmpsAUUS, PhoneO
   where EmpsAUUS.Country=PhoneO.Country;
```

Erklärung:

Der Data Step geht zeilenweise vor.
Findet er in einer Zeile keinen gleichen ***country-Wert*** in beiden Dateien (hier Zeile 3, EmpsAUUS), dann nimmt er den Wert, der noch im Zwischenspeicher PDV von der vorherigen Zeile stand und verbindet diese Zeile (hier: Kylie F) mit allen weiteren Zeilen aus der anderen Datei, die den gleichen country-Wert haben usw.

# Lösung: Data Step: 6 Zeilen

**EmpsAUUS**

| First | Gender | Country |
|---|---|---|
| Togar | M | AU |
| Kylie | F | AU |
| Stacey | F | US |
| Gloria | F | US |
| James | M | US |

**PhoneO**

| Country | Phone |
|---|---|
| AU | +61 (2) 5555-1500 |
| AU | +61 (2) 5555-1600 |
| AU | +61 (2) 5555-1700 |
| US | +1 (305) 555-1500 |
| US | +1 (305) 555-1600 |

Ergebnis:

**EmpsOfc**

| 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 |

Many-to-many Verknüpfung kann beim Data Step zu „sinnfreien“ Ergebnissen führen!

Erklärung

SQL macht kleine kartesische Produkte: jeder mit jedem, wenn die Verknüpfungs-Variablen Werte übereinstimmen

# Lösung: Proc SQL: 12 Zeilen

**EmpsAUUS**

| First | Gender | Country |
|---|---|---|
| Togar | M | AU |
| Kylie | F | AU |
| Stacey | F | US |
| Gloria | F | US |
| James | M | US |

**PhoneO**

| Country | Phone |
|---|---|
| AU | +61 (2) 5555-1500 |
| AU | +61 (2) 5555-1600 |
| AU | +61 (2) 5555-1700 |
| US | +1 (305) 555-1500 |
| US | +1 (305) 555-1600 |

PROC SQL Ergebnis:

**EmpsOfc**

| 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 |

Frage 7

Die neue Datei three sieht folgendermaßen aus:

THREE

| NUM | CHAR1 | CHAR2 |
|---|---|---|
| 1 | A | |
| 2 | B | X |
| 3 | | Y |
| 4 | D | |
| 5 | | V |

# Vorbereitung zur Frage: Merge – Proc SQL

ONE

| NUM | CHAR1 |
|---|---|
| 1 | A |
| 2 | B |
| 4 | D |

TWO

| NUM | CHAR2 |
|---|---|
| 2 | X |
| 3 | Y |
| 5 | V |

The following SAS program is submitted creating the output table THREE:

```
data three;
merge one (in = in1) two (in = in2);
   by num;
run;
```

Frage:
Welcher der vier SQL Codes erzeugt eine äquivalente Datei *three ?*

THREE

| NUM | CHAR1 | CHAR2 |
|---|---|---|
| 1 | A | |
| 2 | B | X |
| 3 | | Y |
| 4 | D | |
| 5 | | V |

A.
```
proc sql;
create table three as
   select *
      from one full join two
      where one.num = two.num;
quit;
```

B.
```
proc sql;
create table three as
   select coalesce(one.num, two.num)
      as NUM, char1, char2
      from one full join two
      where one.num = two.num;
quit;
```

C.
```
proc sql;
create table three as
   select one.num, char1, char2
      from one full join two
      on one.num = two.num;
quit;
```

D.
```
proc sql;
create table three as
   select coalesce(one.num, two.num)
      as NUM, char1, char2
      from one full join two
      on one.num = two.num;
quit;
```

Erklärung

Hinweis:

Im Data-Step darf es eine Spalte mit gleichem Namen gar nicht doppelt geben, jeder Variablenname muss eindeutig sein.

# Lösung: Code D

1. Die coalesce-Funktion sucht den ersten **non-missing** Value und schreibt ihn raus.

Die beiden num-Spalten in beiden Tabellen ist somit nur noch einmal vorhanden als NUM und alle Werte stehen in EINER Spalte.

**Coalesce(one.num, two.num) as NUM**

Gibt es keine Übereinstimmungen, dann wird in die neue Spalte der non-missing Wert geschrieben.

2. Außerdem muss ein **full join** mit on bei der Variablen-Auflisten geschrieben werden, nicht mit where!

   Gleiches gilt für left join, right join und inner join.

Frage 8

Substr-Funktion

Data Step

# Was 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;
```

Erklärung

Beispiele für „Standard“-Längen ausgewählter Funktionen

Aber: || oder !! verbinden auch Werte, sind keine Funktionen: die neue Variable wird so lang wie die einzelnen Werte zusammen.

Keine Standardlänge!

## Funktionen

Wenn neue Variablen mittels Funktionen erstellt werden, kann es sein, dass die Variablen durch diese Funktionen eine „Standard“-Länge erhalten.

| Funktionsname | Länge der neuen Variablen | Beschreibung |
|---|---|---|
| SUBSTR | Wie Originalvariable | Zeichen extrahieren |
| SCAN | Wie Originalvariable | Wörter extrahieren |
| TRANWRD | 200 Bytes | Ersetzen |
| CATS, CATX, CATT, CAT | 200 Bytes | Werte/Spalten verbinden |
| REPEAT | 200 Bytes | Wiederholt einen alphanum. Ausdruck |
| SYMGET | 200 Bytes | Liest zur Laufzeit im DataStep Makrovar aus |

PDV=Program Data Vector

| Alphabetic List of Variables and Attributes | | | |
|---|---|---|---|
| # | Variable | Type | Len |
| 1 | package | Char | 3 |
| 2 | part1 | Char | 3 |
| 3 | part2 | Char | 3 |
| 4 | part3 | Char | 3 |

|| (sog. Pipes) sind Zeichen, um Werte miteinander zu verbinden (oder !!)

## Lösung

In der **Kompilierungsphase** des Data Steps wird die Variable ***package*** mit der Länge von 3Bytes angelegt. Wenn diese Länge einmal im Zwischenspeicher (PDV) ist , dann wird sie nicht mehr geändert!

Variablen, die mit der **Substr-Funktion** erstellt werden, bekommen die Länge der Originalvariablen, also der Variablen, auf die sie sich beziehen.

Hier: ***package*** = 3B, also haben die Variablen part1-part3 auch 3B (jeweils 1 Buchstabe mit 2 nachfolgenden Leerzeichen).

Da zum Schluss wieder die Variable ***package*** genommen wird, die nur 3B Länge hat, ist das Ergebnis nur: S _ _ (S mit 2 nachfolgenden Leerzeichen)!

Frage 9

Output-Anweisung

Wie viele Zeilen hat die Datei ***work.one***, wenn das Programm unten abgeschickt wurde?

| Var1 | Var2 |
|---|---|
| A | one |
| A | two |
| B | three |
| C | four |
| A | five |

Inputdatei

```
data work.one
     work.two;
  set inputdatei;
  if var1='A' then output work.one;
  output;
run;
```

# Lösung: 8 Zeilen

Datei: work.one

| var1 | var2 |
|---|---|
| A | one |
| A | one |
| A | two |
| A | two |
| B | three |
| C | four |
| A | five |
| A | five |

Das OUTPUT in der If-Anweisung filtert insgesamt 3 Zeilen in die Datei ***work.one*** nämlich immer, wenn die Bedingung var=A zutrifft.

Die zweite OUTPUT Anweisung schreibt jede Zeile aus der *inputdatei* in die Dateien raus – und zwar in beide Dateien. ***Work.one*** hat 8 Zeilen und ***work.two*** hat 5 Zeilen.

# Frage 10

## Wie viele Zeilen hat die Datei ***test***?

Datei: sashelp.class

| 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 |
| Janet | 15 | 62.5 |
| Jeffrey | 13 | 62.5 |
| John | 12 | 59 |
| Joyce | 11 | 51.3 |
| Judy | 14 | 64.3 |
| Louise | 12 | 56.3 |
| Mary | 15 | 66.5 |
| Philip | 16 | 72 |
| Robert | 12 | 64.8 |
| Ronald | 15 | 67 |
| Thomas | 11 | 57.5 |
| William | 15 | 66.5 |

```
Data test;
  Array Numwerte{2} age height;
  If _n_=1 then set sashelp.class (where=(name='Alfred'));
Run;
```

# Lösung: 2 Zeilen!

Beginn der Ausführung, 1. Iteration: Alle Variablen werden auf Missing initialisiert.

*If _N_=1* ist wahr: Datei wird gelesen.

Bei RUN; wird die Beobachtung in die Datei geschrieben; Rückkehr zum Beginn des Dataschritts.

2. Iteration: keine Reinitialisierung der Variablen, da alle Variablen aus der SAS Datei kommen.

Jetzt ist *If _N_=1* falsch.

Bei RUN; wird die Beobachtung, die von der ersten Iteration noch im PDV ist (da keine Reinitialisierung der Variablen) in die Datei geschrieben.

Ende

| age | height | weight | Name | Sex |
|---|---|---|---|---|
| 14 | 69 | 112.5 | Alfred | M |
| 14 | 69 | 112.5 | Alfred | M |

# Makro-Fragen

Frage 11

# Wie kann man den Text in der %PUT Anweisung einrücken?

```
891  %macro name(fullname);
892     %let first=%qscan(&fullname,2);
893     %let last=%qscan(&fullname,1);
894     %let newname=&first &last;
895     %put      &newname;
896  %mend name;
NOTE: The macro NAME completed compilation without errors.
      19 instructions 388 bytes.
897
898  %name(%str(Taylor, Jenna))
Jenna Taylor
```

**Blanks are ignored.**

Maskierungsfunktion (Quoting Functions):

**%nrstr-Funktion:**

Diese maskiert alle Sonderzeichen UND die Makro-Trigger:

& %

Bsp:
%nrstr(Müller&Co)

Bewirkt, dass **&Co** nicht als Makrovariable gedeutet wird

## Lösung: Mit der **%str Funktion**

**Maskierungsfunktionen** heben Zeichen in ihrer Bedeutung auf, z. B.

**%str(;)** dann ist das Semikolon nicht mehr das Ende einer Anweisung, sondern ein beliebiges alphanumerisches Zeichen: **%put %str( ) eingerückter Text;**

Lösung zur Frage:

```
%put %str(                    )&newname;
```

Die %str-Funktion maskiert nicht die Makro-Trigger: **& %**

Frage 12

Call Symput(x)

# Wie werden die Makrovariablen durch Call Symput erstellt Global, lokal, teils-teils oder gar nicht?

```
%macro testfrage;
   data _null_;
     call symput ('test_1', 'Was bin ich?');
   run;

 proc sql noprint;
   create table new as select name from sashelp.class;
 quit;

   data _null_;
      call symput ('Test_2', 'Global oder Lokal?');
   run;

  %put _local_ ;
   %put _user_ ;
%mend testfrage;

%testfrage
```

## Call Symput(x)

Der Proc SQL Schritt zwischen den 2 Data Steps erzeugt eine lokale Symboltabelle, daher wird test_2 in dieser dann erstellt (=closest existing symbol table).

```
TESTFRAGE TEST_2 Global oder Lokal?
GLOBAL SASWORKLOCATION "C:\Users\geraf
Files\_TD6136_GERAFL-2_\Prc2/"
GLOBAL SYS_SQL_IP_ALL -1
GLOBAL SYS_SQL_IP_STMT
GLOBAL TEST_1 Was bin ich?
```

# Lösung: Test_1: global, Test_2: lokal

- A. Beide sind global
- B. Beide sind lokal
- C. Teils-teils: Eine global, die andere lokal
- D. Sie werden nicht erstellt, es gibt eine Fehlermeldung

- **Bei Call Symput sind 2 Regeln wichtig zu wissen:**

- 1. Sucht nach einer **bestehenden** Makrovariablen, mit gleichem Namen. Wenn sie existiert, wird diese benutzt – egal ob lokal oder global.

- 2. Wenn die Ziel**variable** nicht existiert, wird eine erstellt, aber es wird **keine Symboltabelle erstellt**. Stattdessen wird sie in die nächstverfügbare (closest existing) Symboltabelle erstellt.

Frage 13

Makroverarbeitung

# Welche Werte haben die Variablen neueVar und neueVar2?

```
data test;
  call symputx('var4','dog');

  neueVar="&var4";
  neueVar2="&var"||"4";
 run;
```

Makro-Programmfluss

Zuerst arbeitet der Makroprozessor, wenn er die Makro-Trigger findet (% &), danach arbeitet call symputX oder symget zur Laufzeit des Data Steps.

# Lösung: Beide neue Variablen haben **&var4** als Inhalt

Die Lösung wird schrittweise in den nächsten Folien erklärt.

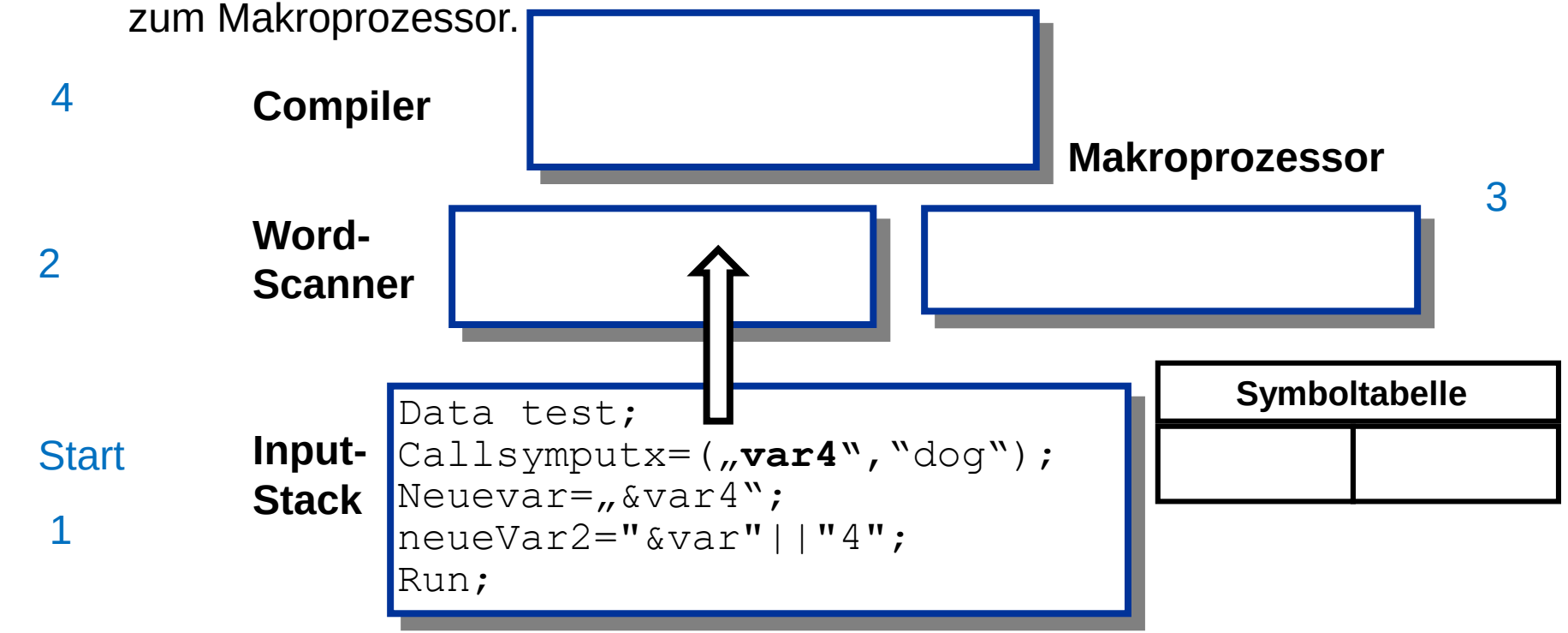
Sie hängt mit den unterschiedlichen Zeitpunkten zusammen, wann ein & aufgelöst wird und wann call symputx verarbeitet wird.

| | Makrovariablen erzeugen | Makrovariablen auslesen |
|---|---|---|
| **Beim Word-Scannen** | %LET | &macvar |
| **Laufzeit** | CALL SYMPUTX | SYMGET('macvar') |

Mit der symget-Funktion würde es klappen **neuvariable=symget(‚var4‘);** Das Ergebnis wäre **dog**

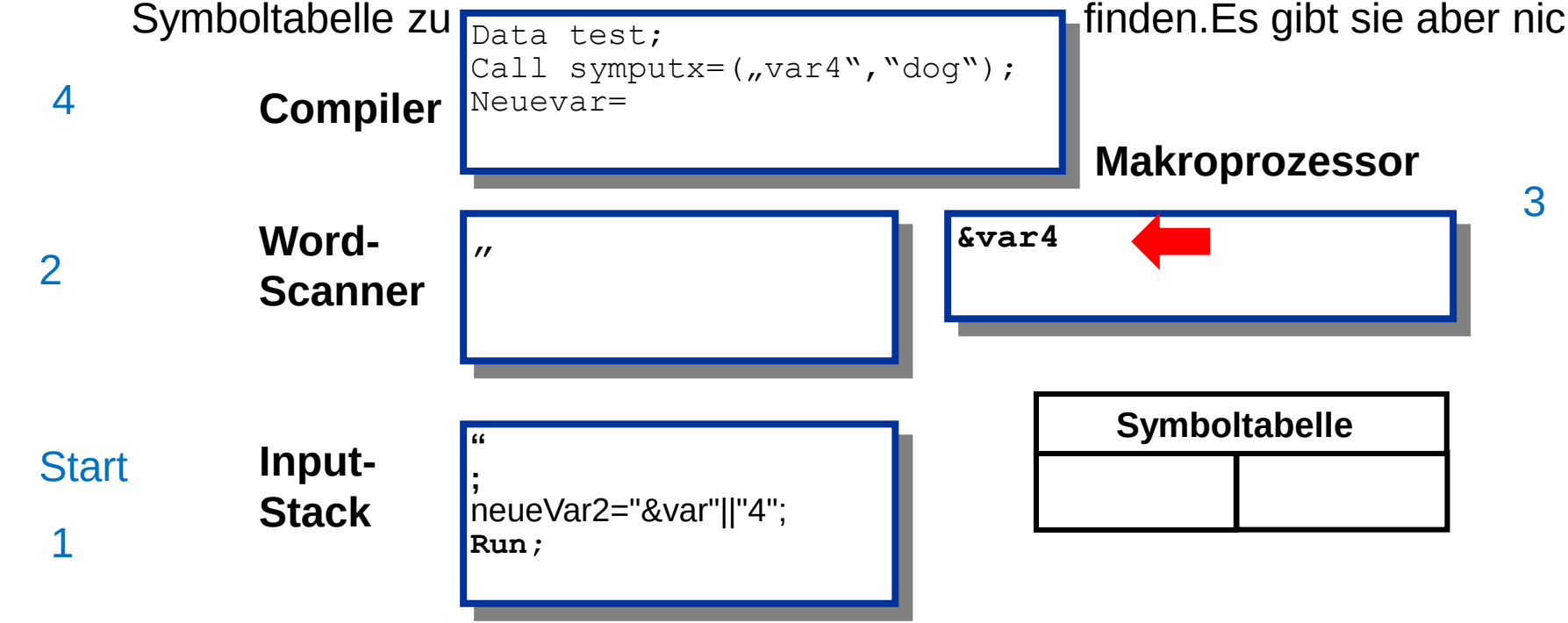
# Makro Verarbeitung - Programmablauf

Egal welche Syntax, alles startet im sog. Input Stack! Von dort geht es an den Word Scanner, der prüft, ob es Makro-Trigger gibt (% &), denn dann geht es zum Makroprozessor.



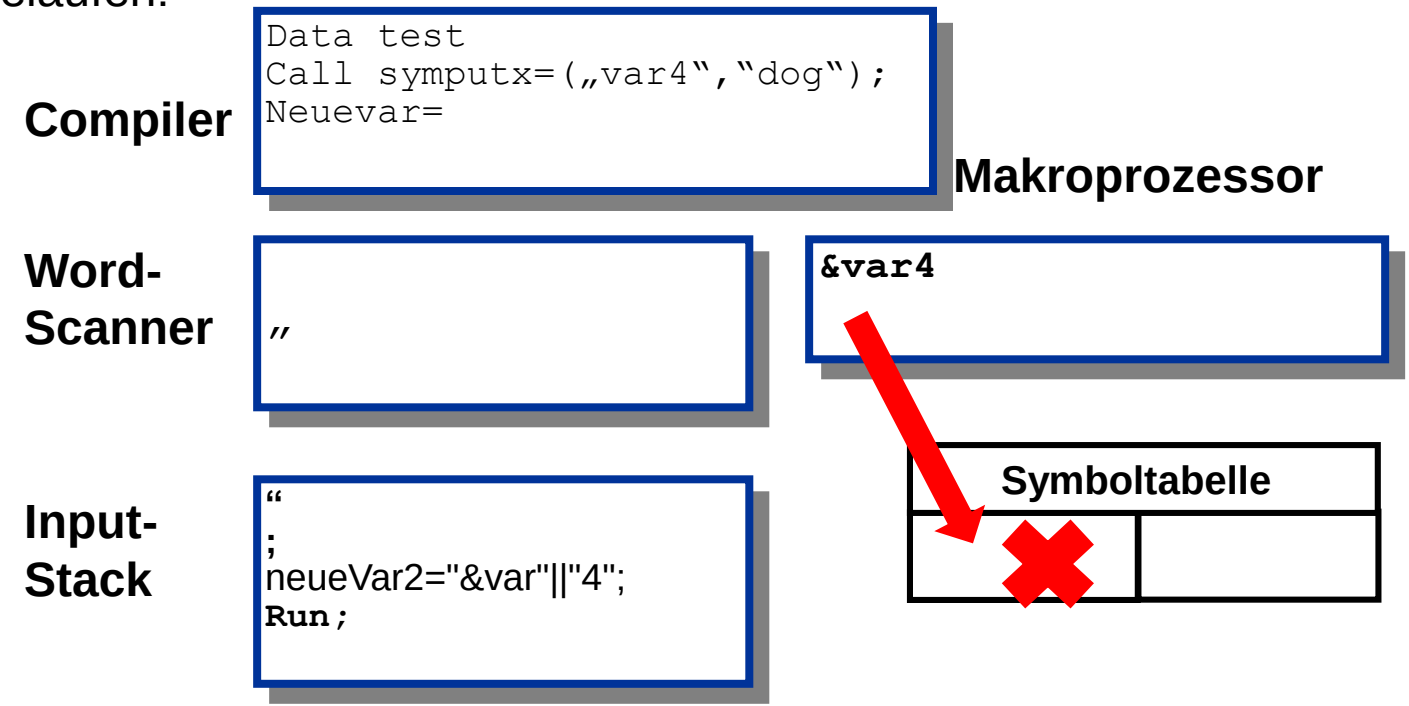
# Makro Verarbeitung - Programmablauf

Der Wordscanner erkennt in den doppelten Anführungszeichen das **&**var4, schickt es an den Makroprozessor und dieser versucht die Makrovariable in der Symboltabelle zu finden.Es gibt sie aber nicht!



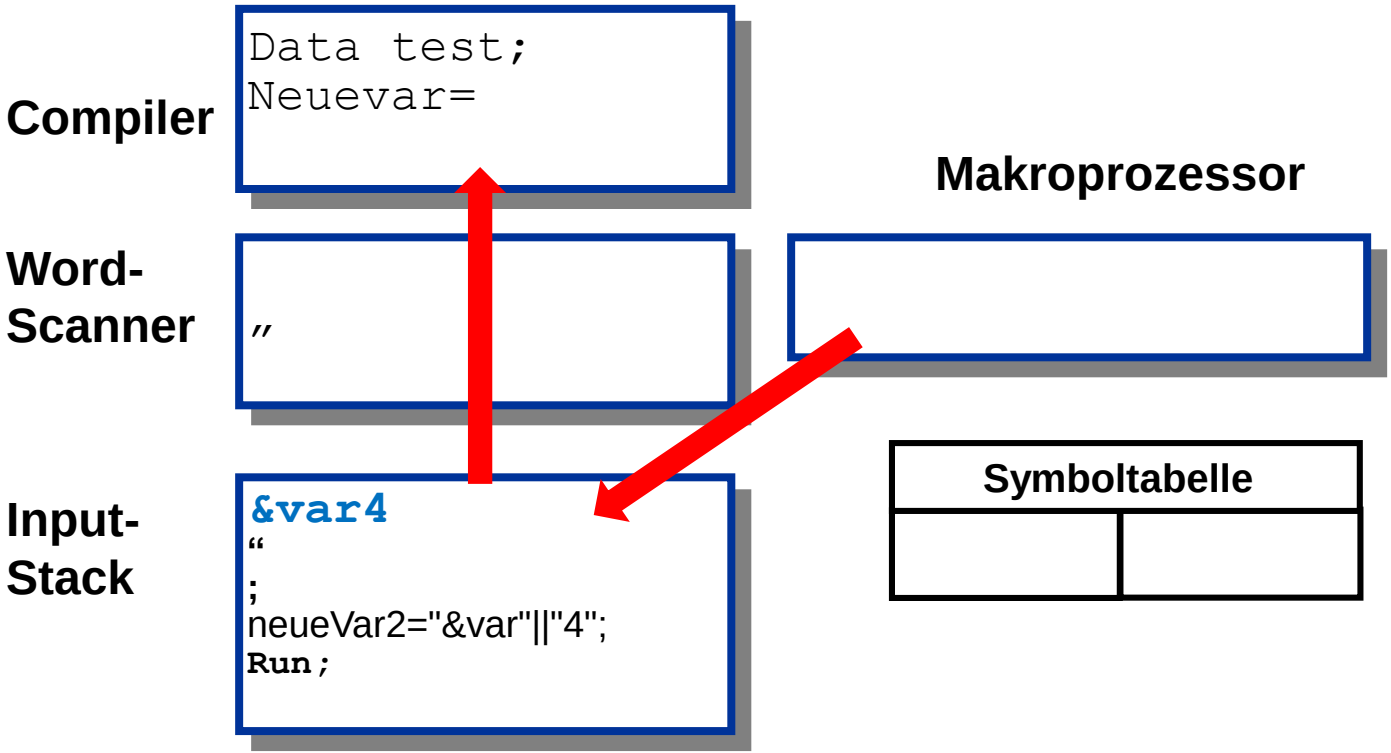
# Makro Verarbeitung - Programmablauf

Der Makroprozessor findet die Makrovariable &var4 nicht, weil sie ja erst beim **run;** im Data Step **zur Laufzeit** erzeugt wird! Bis jetzt ist der Data Step ja noch nicht gelaufen!



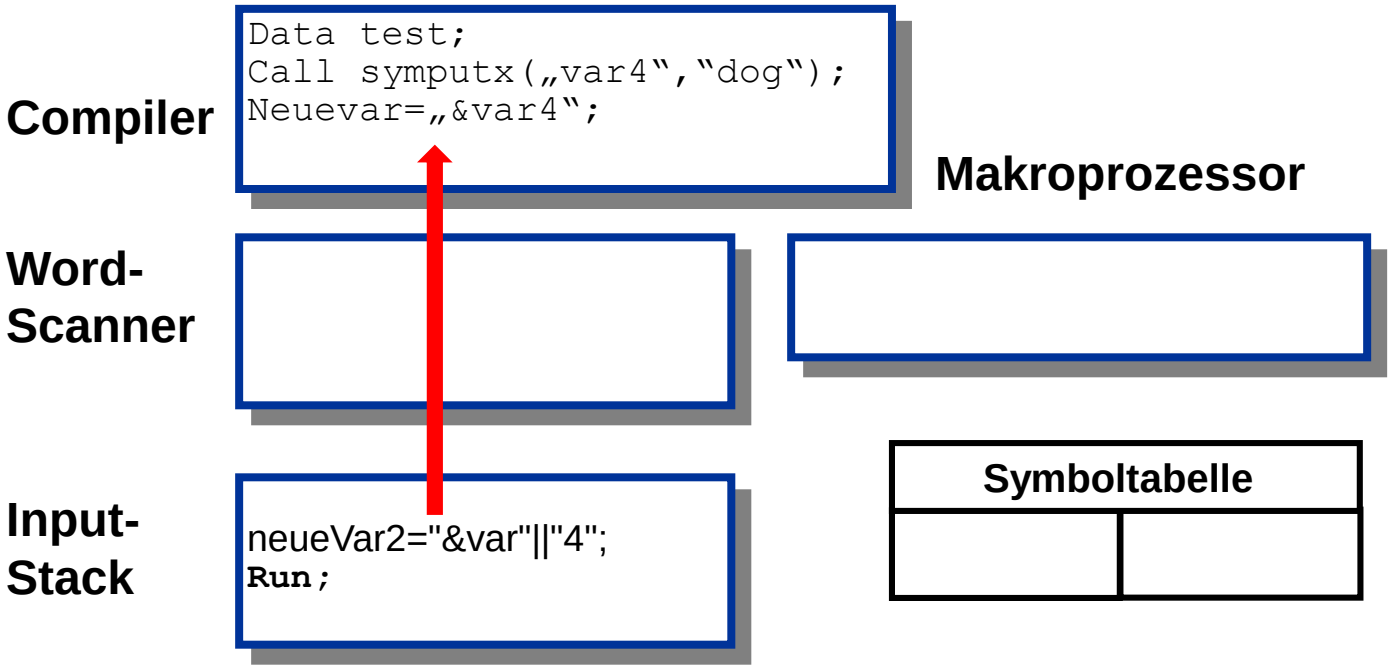
# Makro Verarbeitung - Programmablauf

Da die Makrovariable nicht gefunden wurde, geht sie wieder in den Input Stack und von dort – so wie sie ist - &var4 – in den Compiler.



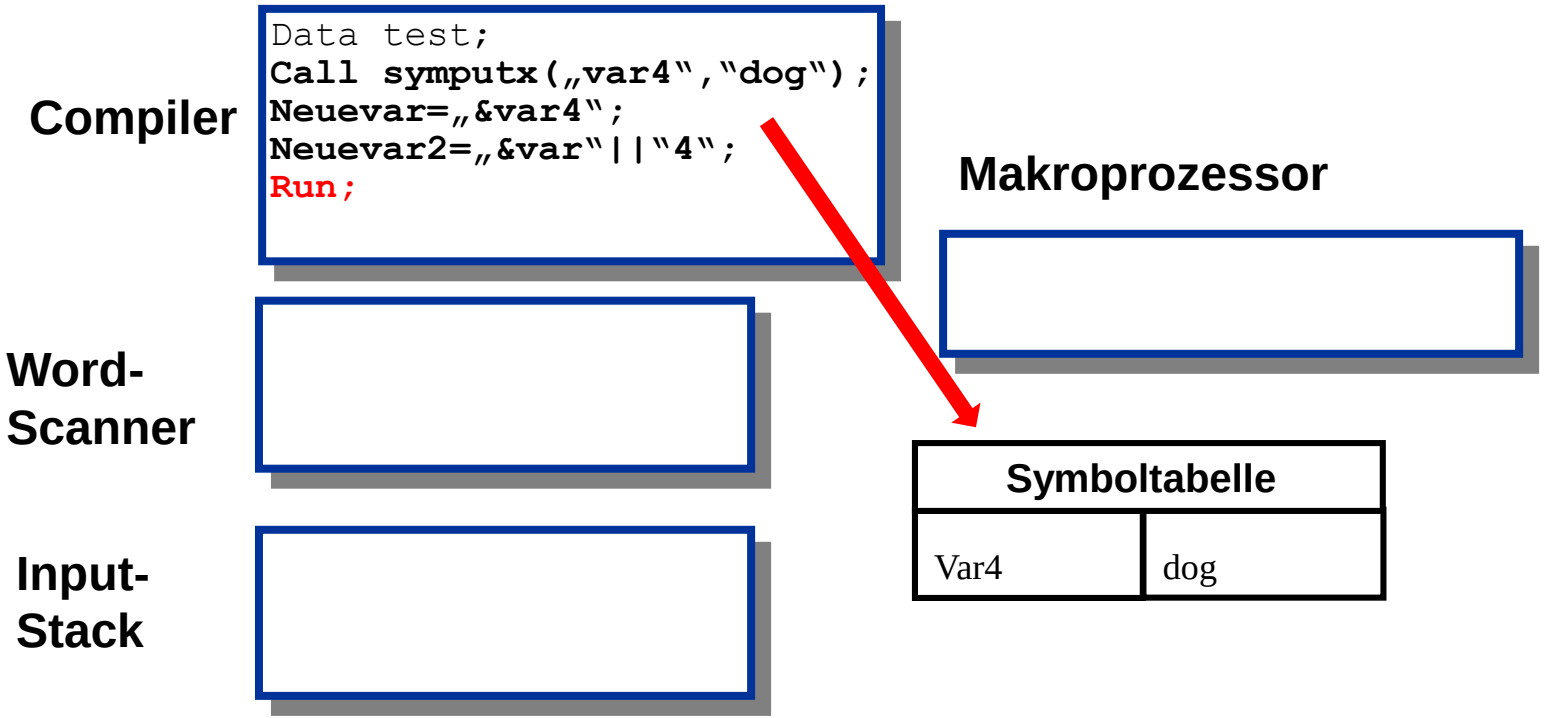
# Makro Verarbeitung - Programmablauf

Der Makroprozessor arbeitet zeitlich früher bevor der Data Step gelaufen ist.



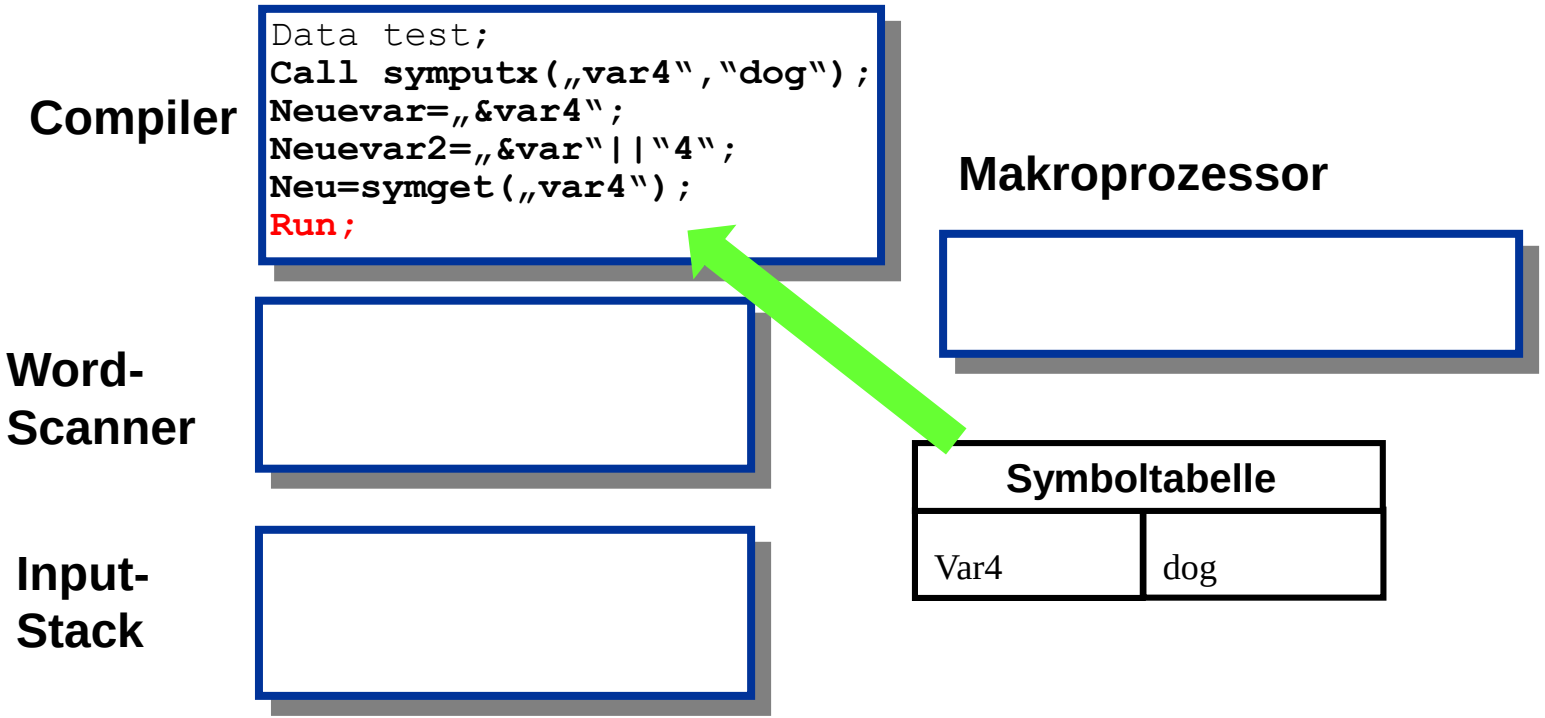
# Makro Verarbeitung - Programmablauf

Erst zur **Laufzeit** des Data Steps arbeitet call symput(x) und schreibt die Makrovariable mit Wert in die Symboltabelle.



# Makro Verarbeitung - Programmablauf

Aber: Mit der Funktion Symget könnte man eine Variable, die mit call symputx erstellt wurde, zur Laufzeit auslesen. Das geht!



Frage 14

Arbeitet die MAKRO-Funktion %substr genauso wie die substr-Funktion?

# Makro-Funktion: %substr: Was ist der Wert von Zeichen2?

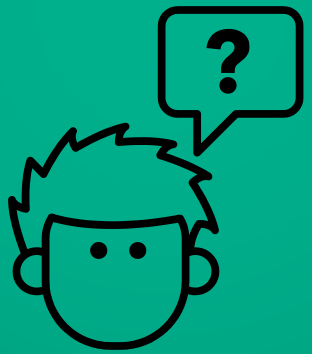
```
%let Wert="ABCDE";
%let Zeichen2=%substr(&WERT,2,1);

%put das Ergebnis ist &=Zeichen2;
```

# Lösung:  Zeichen2=A  (nicht B!)

Ja, die Makro-Funktion %substr arbeitet genauso.

**Aber**: Makro ist immer Text, daher gehören die Anführungszeichen zum Wert dazu und sind das Zeichen1, Zeichen2 ist dann das **A!**

# Fragen?



# Vielen Dank für Ihre Teilnahme!


Lernen Sie unser kostenloses E-Learning kennen! Unsere Trainer stehen Ihnen währenddessen zur Verfügung, um Sie zu begleiten und zu unterstützen.

http://www.sas.com/de_de/training/courses/connected-elearning.html

# Folien zum Download unter www.sas.de/lunchtime

Besten Dank für Ihr Feedback ☺

sas.com