The George Washington University
Department of Statistics

STAT 4197/6197 - Fall 2019

Week 3 – September 13, 2019

Major Topic: Working with Formats and Informats, and Transforming Data

Detailed Topics:

1. User-Defined Formats
    a. Creating, Storing, Accessing, and Maintaining Formats
    b. Grouping Data Values Using Formats
    c. Removing Formats, and Labels from SAS Data Sets

2. Picture Formats and User-Defined Informats

3. Transforming Data
    a. Creating New Variables
        i. by Recoding Values
        ii. with an expression
    b. SELECT/WHEN/OTHERWISE Statements
    c. IFC/IFN Functions
    d. Case Expression in PROC SQL
    e. SUM and RETAIN Statements

Readings:

1. Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012

2. Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015

3. Cody, R. Cody's Data Cleaning Techniques Using SAS®, Third Edition - March 2017

4. The SELECT Statement in the SAS DATA Step

Business Scenario

Display country names instead of country codes in a report.

Current Report (partial output)

Obs	Employee_ID	Salary	Country	Birth_ Date	Hire_ Date
1	120102	$108,255	AU	AUG1973	JUN1993
2	120103	$87,975	AU	JAN1953	JAN1978
3	120121	$26,600	AU	AUG1948	JAN1978

Desired Report (partial output)

Obs	Employee_ID	Salary	Country	Birth_ Date	Hire_ Date
1	120102	$108,255	Australia	AUG1973	JUN1993
2	120103	$87,975	Australia	JAN1953	JAN1978
3	120121	$26,600	Australia	AUG1948	JAN1978

p105d02

18

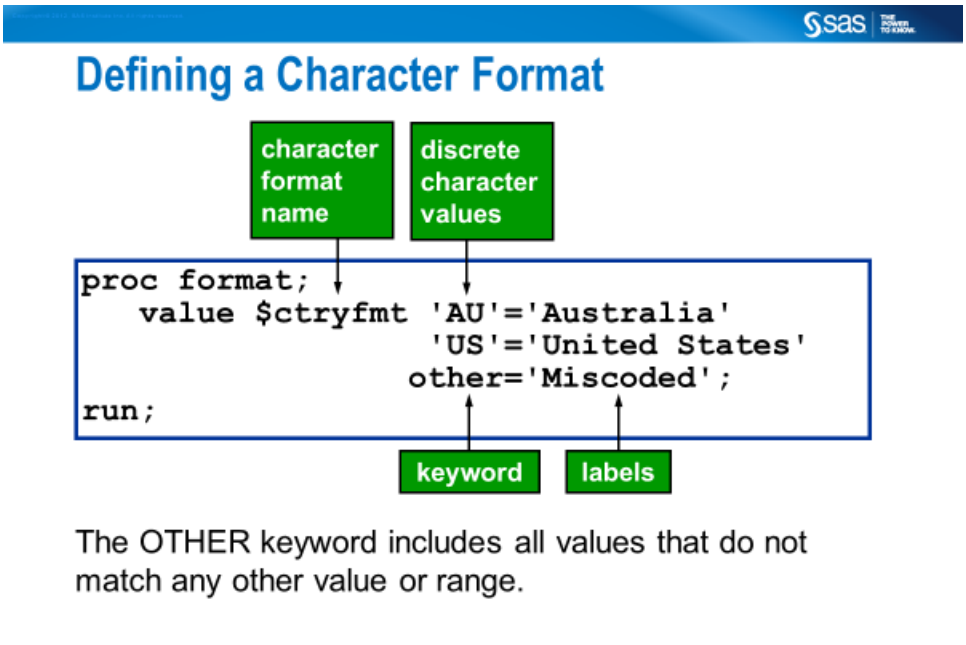
Business Scenario

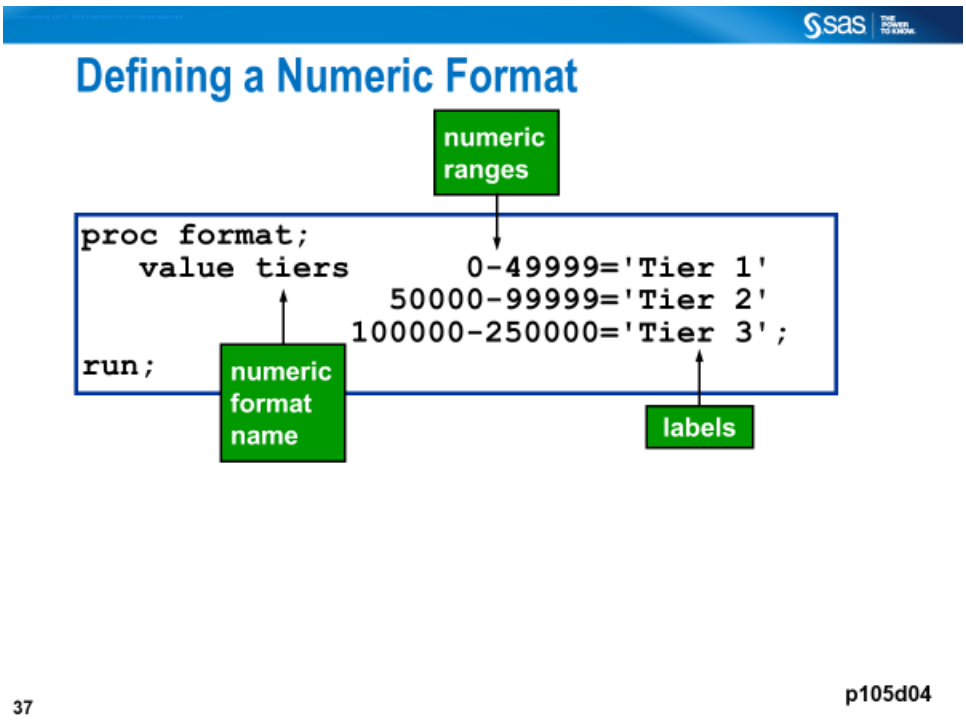
Management has requested that country names, instead of country codes, be used in reports.

Country	Population	Country_ID
AU	20,000,000	160
CA	.	260
DE	80,000,000	394
IL	5,000,000	475

Country	Population	Country_ID
Australia	20,000,000	160
Canada	.	260
Germany	80,000,000	394
Israel	5,000,000	475

3





The FORMAT procedure enables you to define your own formats for variable values. Formats determine how variable values are printed.

VALUE Statement

```
VALUE format-name range1='label '
                  range2='label '
                  . . . ;
```

A format name

- can be up to 32 characters in length
- for character formats, must begin with a dollar sign ($), followed by a letter or underscore
- for numeric formats, must begin with a letter or underscore
- cannot end in a number
- cannot be given the name of a SAS format
- cannot include a period in the VALUE statement.

22

Applying a Format

User-defined and SAS formats can be applied in a single FORMAT statement.

```
proc print data=orion.sales label;
   var Employee_ID Job_Title Salary
       Country Birth_Date Hire_Date;
   format Salary dollar10.
          Birth_Date Hire_Date monyy7.
          Country $ctryfmt.;
run;
```

- A period (for example, at the end of the $CTRYFMT format) is required when user-defined formats are used in a FORMAT statement.

p105d03

T 27

Viewing the Output

Partial PROC PRINT Output

Obs	Employee_ID	Salary	Country	Birth_ Date	Hire_ Date
1	120102	$108,255	Australia	AUG1973	JUN1993
2	120103	$87,975	Australia	JAN1953	JAN1978
3	120121	$26,600	Australia	AUG1948	JAN1978
4	120122	$27,475	Australia	JUL1958	JUL1982
5	120123	$26,190	Australia	SEP1968	OCT1989

21

VALUE Statement

```
VALUE format-name range1='label '
                  range2='label '
                  . . . ;
```

Each range can be

- a single value
- a range of values
- a list of values.

Labels

- can be up to 32,767 characters in length
- are enclosed in quotation marks.

23

Single Values

On the left side of the equal sign, you can have single values. Character values should be enclosed in quotation marks.

case-sensitive

```
proc format;
   value $gender 'F'   = 'Female'
                 'M'   = 'Male'
                other = 'Miscoded';
   value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
```

You can have multiple single values with commas separating the values.

```
value $gender 'F','FEM','FEMALE' = 'Female'
              'M','MAL','MALE'   = 'Male';
```

61

Ranges

On the left side of the equal sign, you can have ranges.

```
proc format;
   value $gender 'F'   = 'Female'
                 'M'   = 'Male'
                other = 'Miscoded';
   value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
```

For character ranges, each string should be enclosed in quotation marks (example: `'A' - 'Z'`).

64

User-Defined Format Example

Ranges can be specified using lists, ranges, discrete values, and keywords.

```
proc format;
   value mnthfmt 1,2,3='Qtr 1'
                 4-6='Qtr 2'
                 7-9='Qtr 3'
                10-12='Qtr 4'
                    .='missing'
                other='unknown';
run;
```

37

Keywords

- OTHER matches all values that do not match any other value or range.
- LOW encompasses lowest possible value.
  LOW does not include missing for numeric variables.
  LOW does include missing for character variables.
- HIGH encompasses highest possible value.
- LOW - HIGH encompasses all values.

```
proc format;
   value $gender 'F'   = 'Female'
                 'M'   = 'Male'
               other = 'Miscoded';
   value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
```

70

Multiple User-Defined Formats

Multiple VALUE statements can be included in a single PROC FORMAT step.

```
proc format;
   value $ctryfmt   'AU'='Australia'
                    'US'='United States'
                  other='Miscoded';

   value tiers      low-<50000 ='Tier 1'
                 50000-<100000='Tier 2'
                 100000-high  ='Tier 3';
run;
```

38 p105d07

Nesting Formats

In the VALUE statement, you can specify that the format use a second format as the formatted value.

Enclose the format name in square brackets:

```
proc format library=orion.MyFmts;
   value $extra ' '='Unknown'
                other=[$country30.];
run;
```

value=[*existing-format*]

18 p210d01

Applying a Numeric Format

Part 1

```
proc format;
   value tiers     low-<50000 ='Tier 1'
                  50000-<100000='Tier 2'
                  100000-high  ='Tier 3';
run;
```

Part 2

```
proc print data=orion.sales;
   var Employee_ID Job_Title Salary
       Country Birth_Date Hire_Date;
   format Birth_Date Hire_Date monyy7.
          Salary tiers.;
run;
```

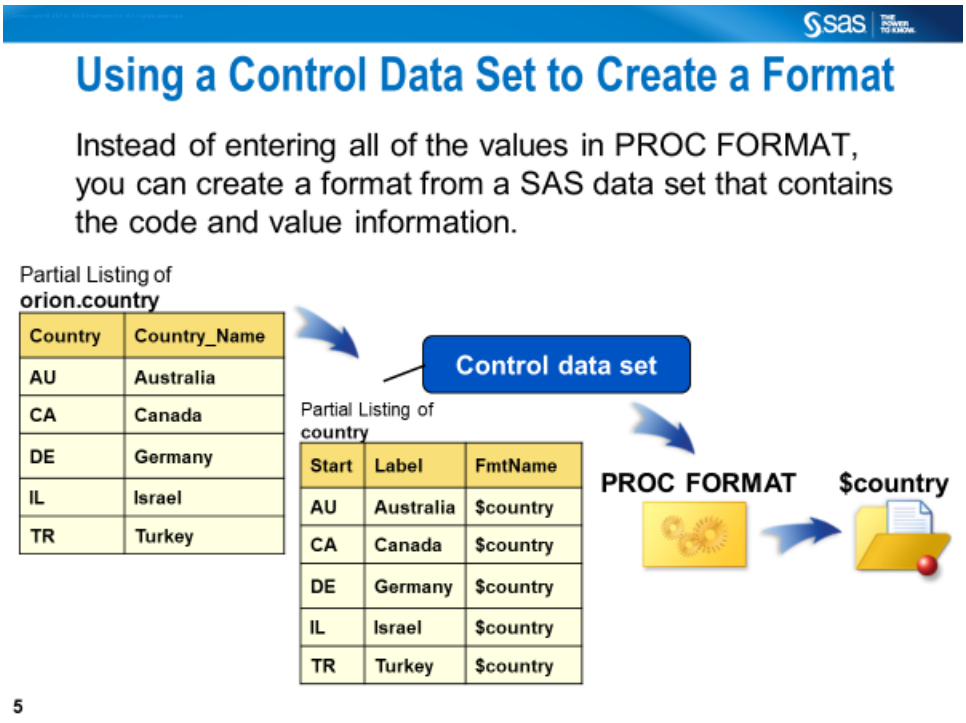
35 p105d06

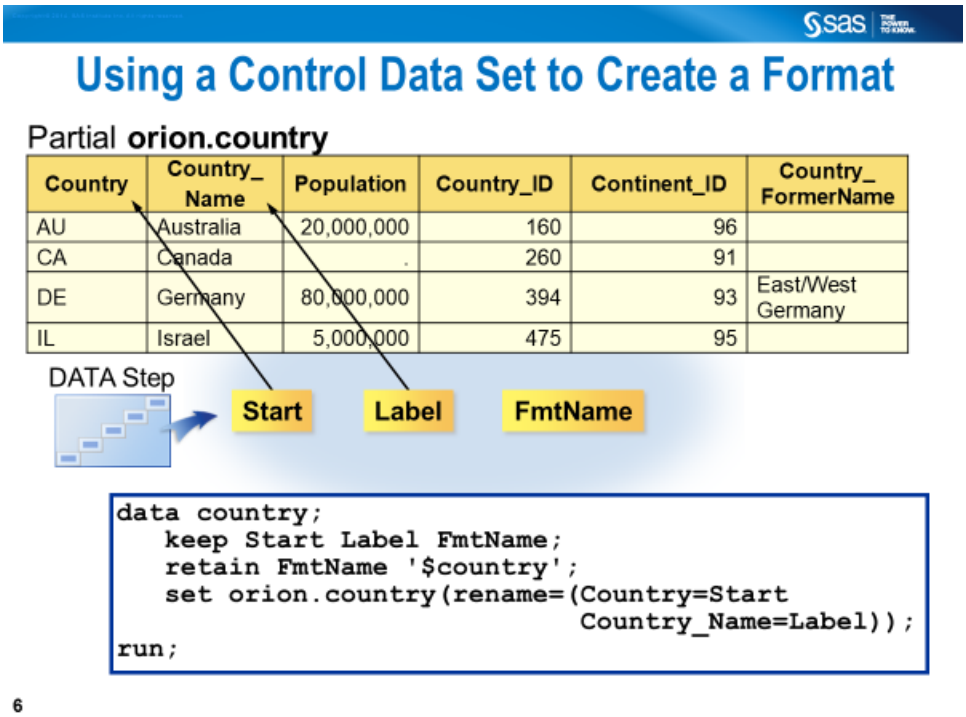
Viewing the Output

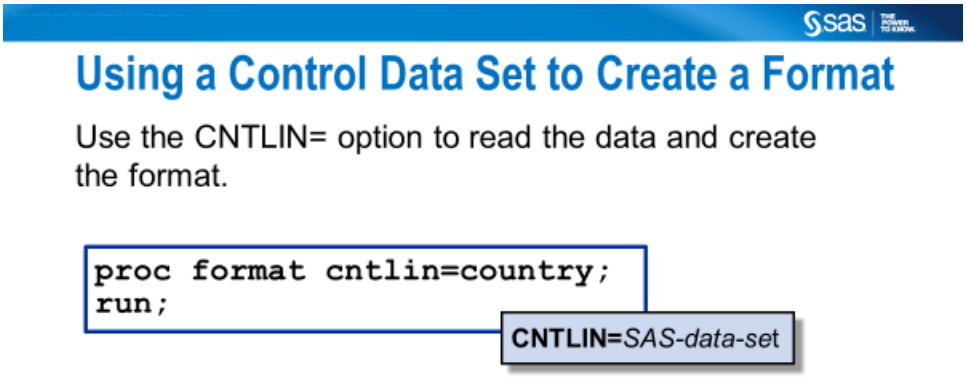
Partial PROC PRINT Output

Obs	Employee_ID	Job_Title	Salary	Country	Birth_ Date	Hire_ Date
1	120102	Sales Manager	Tier 3	AU	AUG1973	JUN1993
2	120103	Sales Manager	Tier 2	AU	JAN1953	JAN1978
3	120121	Sales Rep. II	Tier 1	AU	AUG1948	JAN1978
4	120122	Sales Rep. II	Tier 1	AU	JUL1958	JUL1982
5	120123	Sales Rep. I	Tier 1	AU	SEP1968	OCT1989

36



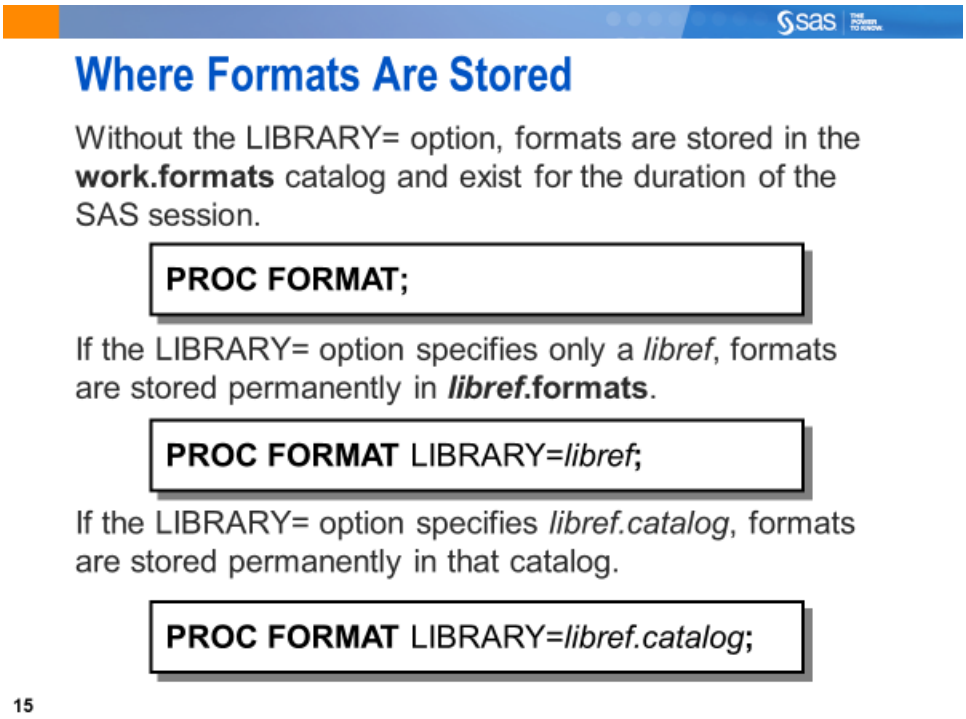




See Cody's book - pages 33-36 on "Creating Format from SAS Data Set" for details.

GitHub\SASGateway\SAScourse:

Ex4_Value_cntlin_compared.sas



- User-written formats can be stored in catalogs.
- The storage location for formats/catalogs can be either temporary or permanent.

Using the LIBRARY= Option

If the LIBRARY= option specifies only a libref, formats are stored permanently in a catalog named **formats**, referenced by ***libref*.formats**.

```
proc format library=orion;
```

PROC FORMAT LIBRARY=*libref*;

13

Using the LIBRARY= Option

If the LIBRARY= option specifies *libref.catalog*, formats are stored permanently in that catalog.

```
proc format library=orion.MyFmts;
```

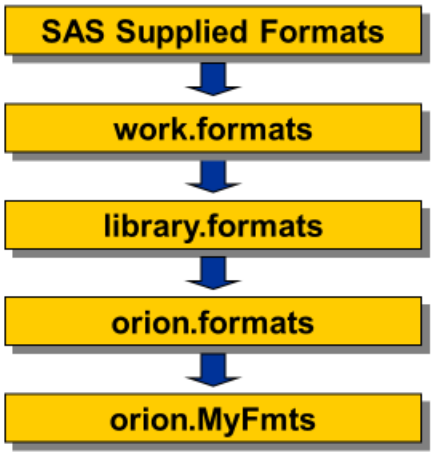
PROC FORMAT LIBRARY=*libref.catalog*;

- Store frequently used formats in permanent catalogs.

14

Using the FMTSEARCH= System Option

```
options fmtsearch=(orion orion.MyFmts);
```



29

CATALOG Procedure

The CATALOG procedure manages entries in SAS catalogs.

```
proc catalog cat=orion.MyFmts;
   contents;
quit;
```

PROC CATALOG CATALOG=*libref.catalog*;
CONTENTS;
QUIT;

Contents of Catalog ORION.MYFMTS

#	Name	Type	Create Date	Modified Date
1	DATES	FORMAT	29Jan08:16:26:39	29Jan08:16:26:39
2	COUNTRY	FORMATC	29Jan08:16:33:30	29Jan08:16:33:30
3	COUNTRY_NAME	FORMATC	20Apr09:15:30:14	20Apr09:15:30:14
4	EXTRA	FORMATC	20Apr09:15:30:14	20Apr09:15:30:14

p210d01

16

Documenting Formats

You can use the FMTLIB option in the PROC FORMAT statement to document the format.

```
proc format library=orion.MyFmts fmtlib;
   select $country;
run;
```

FORMAT NAME: $COUNTRY LENGTH: 13 NUMBER OF VALUES: 7
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 13 FUZZ: 0

START	END	LABEL (VER. V7|V8 05MAY2009:12:34:42)
AU	AU	Australia
CA	CA	Canada
DE	DE	Germany
IL	IL	Israel
TR	TR	Turkey
US	US	United States
ZA	ZA	South Africa

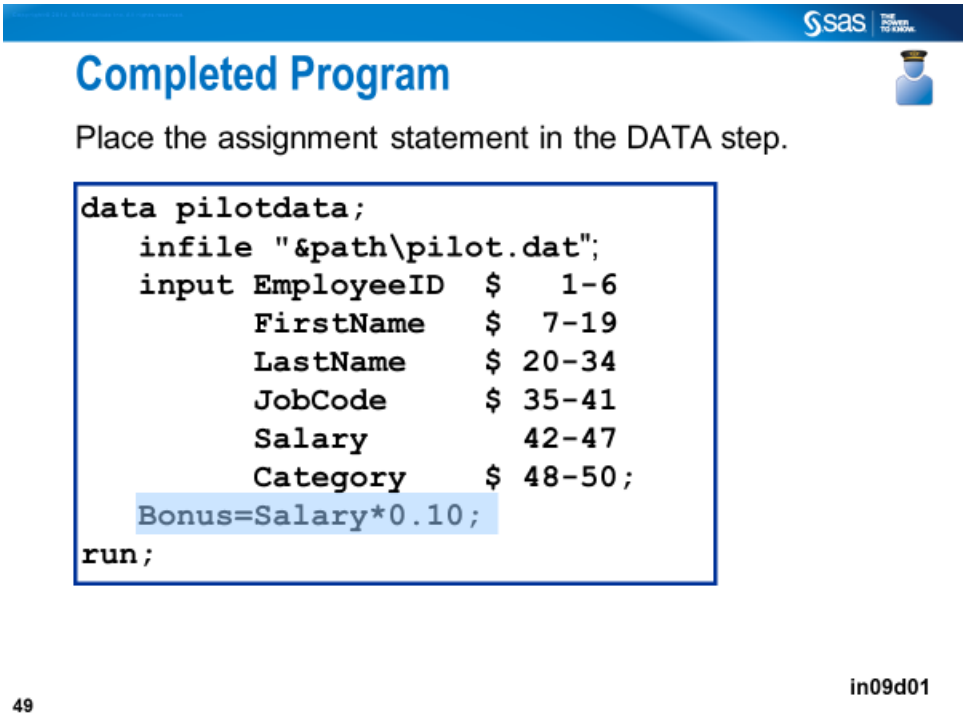
p210d01

17

Picture Statement in the FORMAT Procedure

With the picture statement, you can	Options with the Picture Statement
• Display numbers with leading zeros • Fill numbers with special characters • Insert message characters into numbers. • Customizing a date, time, or date-time display • Format currency when there is no SAS format available	NOEDIT= Specify that numbers are message characters rather than digit selectors PREFIX= Specify a character prefix for the formatted value FILL= Specify a character that completes the formatted value MULT= Specify a number to multiply the variable's value by before it is formatted Source: SAS® Documentation

Creating New Variables using the Assignment Statement



Assignment Statement

The assignment statement evaluates an expression and stores the result in a new variable or an existing variable.

Examples:

```
      name = 'Jane Doe';
   revenue = 157900;
      date = '10MAY2007'd;
     total = price * quantity;
    cityst = city !! state;
   product = upcase(product);
   average = mean(jan, feb, mar);
```

variable

expression

Assignment statements evaluate the expression on the right side of the equal sign and store the result in the variable that is specified on the left side of the equal sign.

3

Expression

The *expression* is a sequence of operands and operators that form a set of instructions that produce a value.

- *Operands* are
  - constants (character or numeric)
  - variables (character or numeric).
- *Operators* are
  - symbols that represent an arithmetic calculation or concatenation
  - a SAS function.

6

Operands

A *constant* is a number or a character string that indicates a fixed value.

```
        name = 'Jane Doe';
     revenue = 157900;
        date = '10MAY2007'd;
```

Character constants must be enclosed in quotation marks.

Character constants are enclosed in quotation marks, but names of variables are not enclosed in quotation marks.

```
       total = price * quantity;
      cityst = city !! state;
     product = upcase(product);
     average = mean(jan, feb, mar);
```

7

Operators

- Arithmetic operators indicate that an arithmetic calculation is performed.
- A concatenation operator concatenates character values.
- A SAS function performs a computation or system manipulation on arguments and returns a value.

```
       total = price * quantity;
      cityst = city !! state;
     product = upcase(product);
     average = mean(jan, feb, mar);
```

8

Arithmetic Operators

Possible arithmetic operators:

Symbol	Definition
**	exponentiation
*	multiplication
/	division
+	addition
-	subtraction

- If a missing value is an operand for an arithmetic operator, the result is a missing value.

9

Determine the Expression

Expressions can contain ***operators*** and ***parentheses***.

Symbol	Definition	Example
+	Addition	CurrentRate+10.27
-	Subtraction	gross_pay-tax
*	Multiplication	price*quantity
/	Division	Minutes/60
( )	Grouping	(jan+feb+mar)/3
**	Exponentiation	x**2

20

Example

```
data newprice;
   set golf.supplies;
   saleprice = price * 0.75;
   saletype = '25% off';
   format price saleprice dollar8.2;
run;
```

VIEWTABLE: Golf.Supplies

	mfg	type	price
1	Crew	Distance	8.1
2	Crew	Spin	8.25
3	Crew	Titanium	9.5
4	Hi-fly	X12000	13.75
5	Hi-fly	X22000	14.6
6	White	Strata	10.6
7	White	Aero	12.3
8	White	XL	14.5
9	White	Flite	16.2

VIEWTABLE: Work.Newprice

	mfg	type	price	saleprice	saletype
1	Crew	Distance	$8.10	$6.08	25% off
2	Crew	Spin	$8.25	$6.19	25% off
3	Crew	Titanium	$9.50	$7.13	25% off
4	Hi-fly	X12000	$13.75	$10.31	25% off
5	Hi-fly	X22000	$14.60	$10.95	25% off
6	White	Strata	$10.60	$7.95	25% off
7	White	Aero	$12.30	$9.23	25% off
8	White	XL	$14.50	$10.88	25% off
9	White	Flite	$16.20	$12.15	25% off

10

Example

```
data newprice;
   set golf.supplies;
   saleprice = price * 0.75;
   saletype = '25% off';
   format price saleprice dollar8.2;
run;
```

Which statement is ***true*** concerning the new variables?

A. **saleprice** and **saletype** are numeric (8 bytes).

B. **saleprice** and **saletype** are character (7 bytes).

C. **saleprice** is numeric (8 bytes) and **saletype** is character (7 bytes).

D. **saleprice** is numeric (8 bytes) and **saletype** is character (8 bytes).

12

IF-THEN / ELSE Statements

- The IF-THEN statement executes ***a statement*** for observations that meet specific conditions.
- The optional ELSE statement gives an alternative action if the THEN clause is not executed.

```
data newprice;
   set golf.supplies;
   if mfg='Crew' then saleprice=price*0.75;
   else if mfg='Hi-fly' then saleprice=price*0.70;
   else if mfg='White' then saleprice=price*0.90;
   format price saleprice dollar8.2;
run;
```

22

Expression

```
if mfg='Crew' then saleprice=price*0.75;
```

expression

- The expression is any valid SAS expression and is a required argument.
- SAS evaluates the expression in an IF-THEN statement to produce a result that is either nonzero, zero, or missing.
- A nonzero and nonmissing result causes the expression to be true; a result of zero or missing causes the expression to be false.

23

Statement

```
if mfg='Crew' then saleprice=price*0.75;
```

statement

The statement can be any executable SAS statement.

Examples:

```
status = 'Unknown'
total = sum(num1, num2, num3)
anniversary = '15AUG2006'd
```

```
count + 1
delete
output
```

26

ELSE Statements

```
if mfg='Crew' then saleprice=price*0.75;
else if mfg='Hi-fly' then saleprice=price*0.70;
else if mfg='White' then saleprice=price*0.90;
```

- Using IF-THEN statements ***without*** the ELSE statement causes SAS to evaluate all IF-THEN statements.
- Using IF-THEN statements ***with*** the ELSE statement causes SAS to execute the IF-THEN statements until SAS encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

29

ELSE Statements

```
if mfg='Crew' then saleprice=price*0.75;
else if mfg='Hi-fly' then saleprice=price*0.70;
else saleprice=price*0.90;
```

The final ELSE statement can be coded without an IF-THEN statement to direct all previous false conditions into the final condition.

32

Conditional Statements

Conditional statements can create values for a new variable based on whether a condition is true or false.

IF *condition* THEN *action*;

```
if JobCode='PILOT1'
   then NewSalary=Salary*1.05;
```

26

Completed Program

The correct approach for our program is to use IF-THEN/ELSE IF logic.

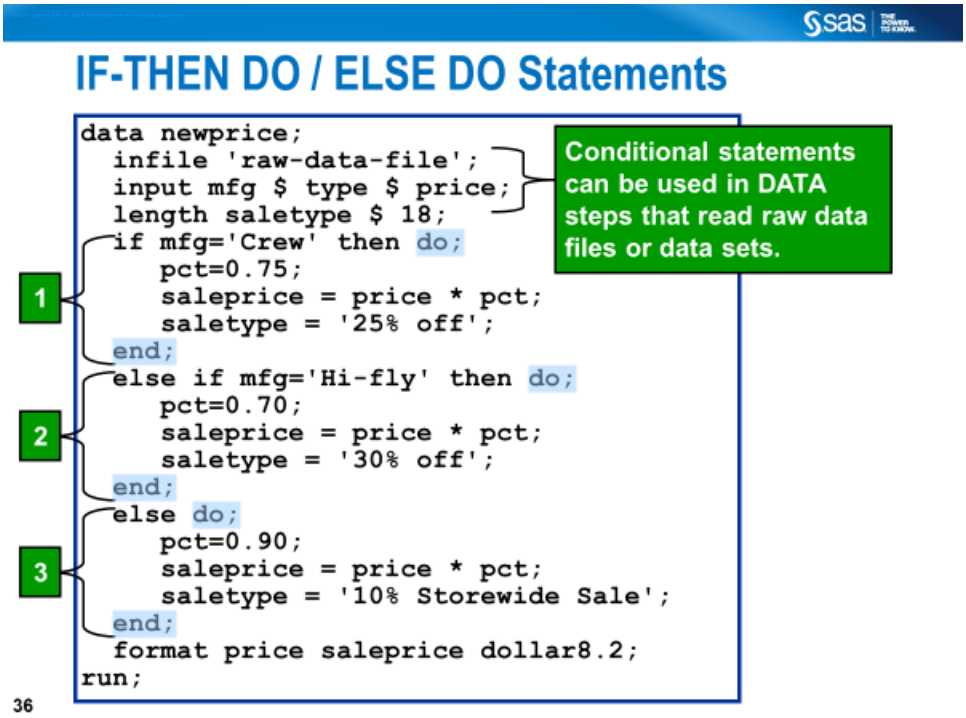
```
data pilotdata;
   infile "&path\pilot.dat";
   input EmployeeID  $   1-6
         FirstName   $  7-19
         LastName    $ 20-34
         JobCode     $ 35-41
         Salary        42-47
         Category    $ 48-50;
   Bonus=Salary*0.10;
   if JobCode='PILOT1' then
      NewSalary=Salary*1.05;
   else if JobCode='PILOT2' then
      NewSalary=Salary*1.07;
   else if JobCode='PILOT3' then
      NewSalary=Salary*1.09;
run;
```

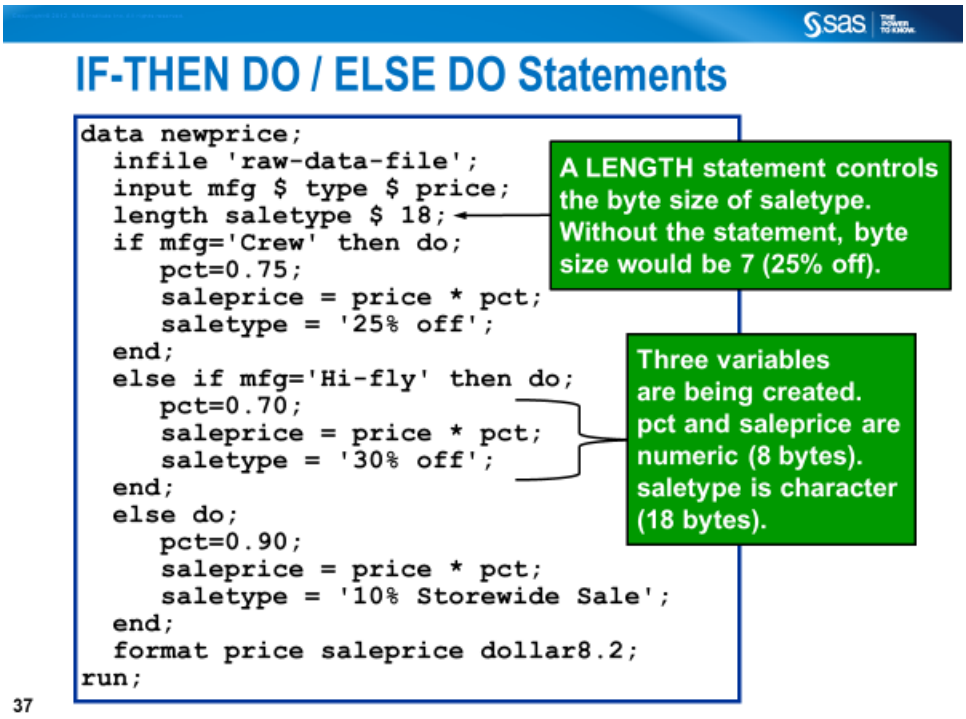
in10d04

42

If you use a variable from the input data source as part of your expression (as we did with the JobCode and Salary Variables), the conditional logic statement must be placed after the INFILE and INPUT statements.

Creating Lookup Values (i.e., New Variables) with the DO statement





SELECT / WHEN / OTHERWISE Statements

An alternative to IF-THEN statements is SELECT / WHEN / OTHERWISE statements.

- The SELECT statement begins a SELECT group.
- SELECT groups contain WHEN statements that identify SAS statements that are executed when a particular condition is true.
- A SELECT group must use at least one WHEN statement.
- An optional OTHERWISE statement specifies a statement to be executed if no WHEN condition is met.
- An END statement ends a SELECT group.

41

SELECT / WHEN / OTHERWISE Statements

IF THEN ELSE

```
data newprice;
   set golf.supplies;
   if mfg='Crew' then
      saleprice=price*0.75;
   else if mfg='Hi-fly' then
            saleprice=price*0.70;
   else if mfg='White' then
            saleprice=price*0.90;
   format price saleprice dollar8.2;
run;
```

SELECT group

```
data newprice;
   set golf.supplies;
   select(mfg);
      when('Crew') saleprice=price*0.75;
      when('Hi-fly') saleprice=price*0.70;
      when('White') saleprice=price*0.90;
   end;
   format price saleprice dollar8.2;
run;
```

42

SELECT / WHEN / OTHERWISE Statements

```
select(mfg);
   when('Crew') saleprice=price*0.75;
   when('Hi-fly') saleprice=price*0.70;
   when('White') saleprice=price*0.90;
end;
```

Based on the partial program above, what is the result if an observation has a value of **mfg='X-treme '**?

```
ERROR: Unsatisfied WHEN clause and no OTHERWISE clause at line
       618 column 3.
mfg=X-treme type=Strata price=$10.60 saleprice=. _ERROR_=1
_N_=10
NOTE: The SAS System stopped processing this step because of
      errors.
```

44

SELECT / WHEN / OTHERWISE Statements

A null OTHERWISE statement prevents SAS from issuing an error message when all WHEN conditions are false.

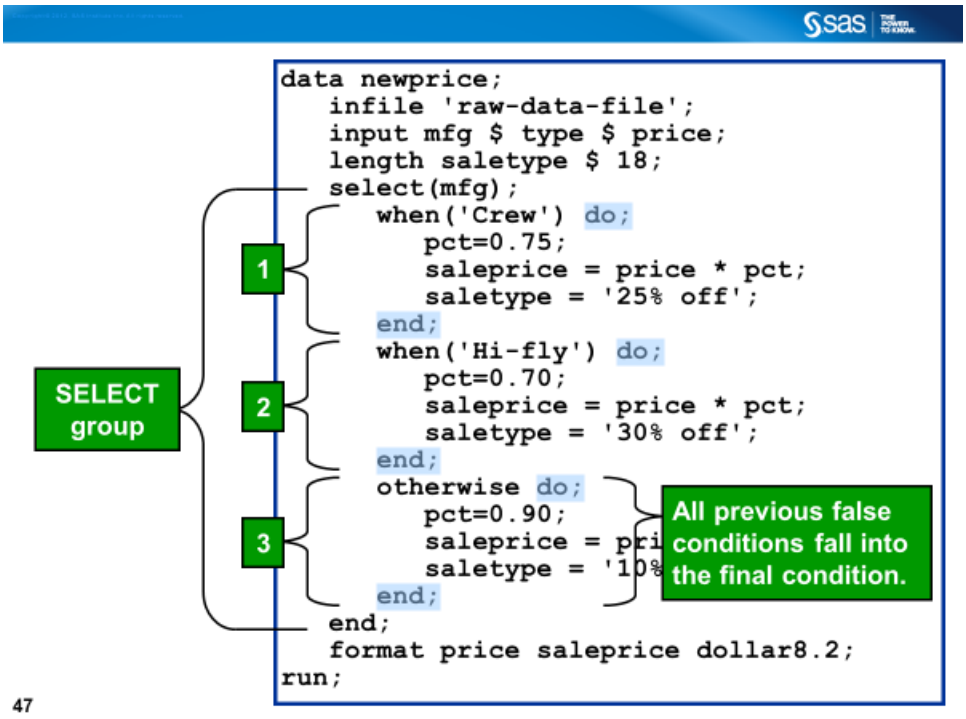
```
ERROR: Unsatisfied WHEN clause and no
OTHERWISE clause at line 618 column 3.
mfg=X-treme type=Strata price=$10.60
saleprice=. _ERROR_=1 _N_=10
```

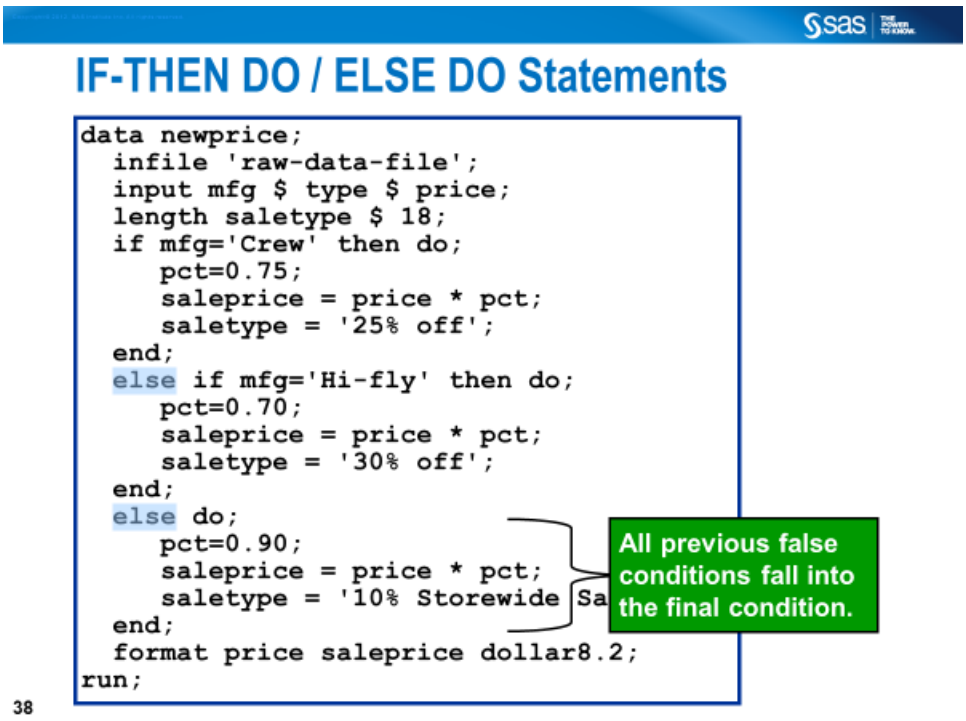
```
select(mfg);
   when('Crew') saleprice=price*0.75;
   when('Hi-fly') saleprice=price*0.70;
   when('White') saleprice=price*0.90;
end;
```

```
select(mfg);
   when('Crew') saleprice=price*0.75;
   when('Hi-fly') saleprice=price*0.70;
   when('White') saleprice=price*0.90;
   otherwise;
end;
```

no ERROR

45





Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

SELECT / WHEN / OTHERWISE Statements

Additional examples:

```
select(payclass);
   when('monthly') amt=salary;
   when('hourly') amt=hrlywage*40;
   otherwise;
end;
```

```
select;
   when(12000<=revenue<=24000) target='goal';
   when(revenue<12000) target='below';
   when(revenue>24000) target='above';
   otherwise;
end;
```

```
select;
   when(mon in ('JUN', 'JUL', 'AUG') and temp>70)
        status='SUMMER';
   when(mon in ('MAR', 'APR', 'MAY'))
        status='SPRING';
   otherwise status='FALL OR WINTER';
end;
```

48

SQL Procedure

- Multiple statements can be included in a PROC SQL step.
- Each statement defines a process and is executed immediately.

```
PROC SQL <option(s)>;
statement(s);
QUIT;
```

6

SELECT Statement Syntax

```
PROC SQL;
SELECT object-item <, ...object-item>
      FROM from-list
      <WHERE sql-expression>
      <GROUP BY object-item <, … object-item >>
      <HAVING sql-expression>
      <ORDER BY order-by-item <DESC>
                  <, …order-by-item>>;
QUIT;
```

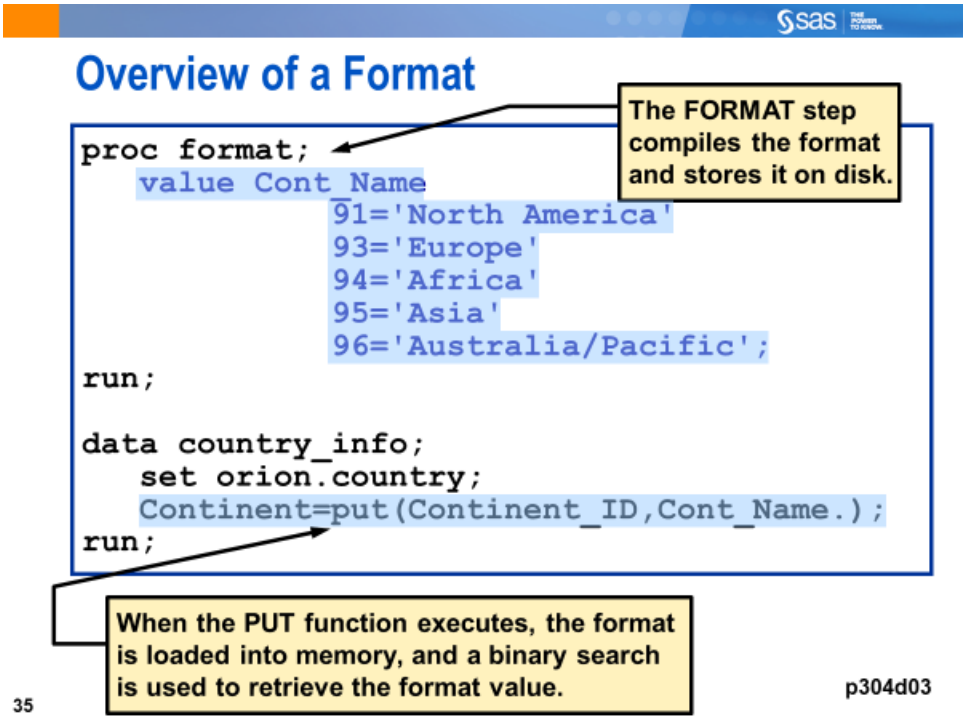
- The specified order of the above clauses within the SELECT statement is required.

12

Some French Women Give Happy Orders.

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

Creating Variables Using PUT Function and Formats



Business Scenario

A manager has asked to see daily sales for April, as well as a month-to-date total for each day for her department.

Partial
orion.aprsales

SaleDate	SaleAmt
01APR2011	498.49
02APR2011	946.50
03APR2011	994.97
04APR2011	564.59
05APR2011	783.01
06APR2011	228.82
07APR2011	930.57

Partial **mnthtot**

SaleDate	SaleAmt		Mth2Dte
01APR2011	498.49		498.49
02APR2011	946.50	=	1444.99
03APR2011	994.97		2439.96
04APR2011	564.59		3004.55
05APR2011	783.01		3787.56

4

4.01 Short Answer Poll – Correct Answer

Open and submit the program in **pdm04a01**. Does this program create the correct values for **Mth2Dte**?

```
data mnthtot;
   set orion.aprsales;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Partial **orion.aprsales**

SaleDate	Sale Amt	Mth2Dte
01APR2011	498.49	.
02APR2011	946.50	.
03APR2011	994.97	.
04APR2011	564.59	.

No, the program creates Mth2Dte with all missing values.

psm04a01

5

Creating an Accumulating Variable

By default, variables created with an assignment statement are initialized to missing at the top of each iteration of the DATA step.

Initialized to missing at the top of the DATA step

```
data mnthtot;
   set orion.aprsales;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Mth2Dte is an example of an accumulating variable that needs to keep its value from one observation to the next.

6

Creating an Accumulating Variable

Retain the values of **Mth2Dte** and set an initial value.

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

RETAIN *variable-name <initial-value> …;*

If you do not supply an initial value, all the values of **Mth2Dte** will be missing.

7 psm04d01

RETAIN Statement: Details

The RETAIN statement

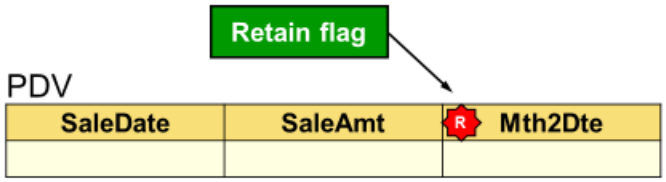
- retains the value of the variable in the PDV across iterations of the DATA step
- initializes the retained variable to missing or a specified initial value before the first iteration of the DATA step
- is a compile-time-only statement.

The RETAIN statement has no effect on variables that are read with SET, MERGE, or UPDATE statements. Variables read from SAS data sets are retained automatically.

8

Compilation: Create an Accumulating Variable

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```



9 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Initialize PDV

PDV

SaleDate	SaleAmt	R Mth2Dte
.	.	0

The input SAS data set must be sorted by **SaleDate** for the program to produce the correct results.

10 ...

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

PDV

SaleDate	SaleAmt	R Mth2Dte
18718	498.49	0

11 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

0 + 498.49

PDV

SaleDate	SaleAmt	R Mth2Dte
18718	498.49	498.49

12 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Implicit OUTPUT;
Implicit RETURN;

PDV

SaleDate	SaleAmt	R	Mth2Dte
18718	498.49		498.49

Write observation to **mnthtot**

13 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Mth2Dte is not reinitialized.

PDV

SaleDate	SaleAmt	R	Mth2Dte
18718	498.49		498.49

14 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

PDV

SaleDate	SaleAmt	R Mth2Dte
18719	946.50	498.49

15 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

498.49 + 946.50

PDV

SaleDate	SaleAmt	R Mth2Dte
18719	946.50	1444.99

16 ...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Implicit OUTPUT;
Implicit RETURN;

PDV

SaleDate	SaleAmt	R Mth2Dte
18719	946.50	1444.99

Write observation to **mnthtot**

17

...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Mth2Dte is not reinitialized.

PDV

SaleDate	SaleAmt	R Mth2Dte
18719	946.50	1444.99

18

...

Execution: Create an Accumulating Variable

SaleDate	SaleAmt
18718	498.49
18719	946.50
18720	994.97
18721	564.59
18722	783.01

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=Mth2Dte+SaleAmt;
run;
```

Continue until EOF

PDV

SaleDate	SaleAmt	R Mth2Dte
18719	946.50	1444.99

19

Creating an Accumulating Variable

```
proc print data=mnthtot noobs;
   format SaleDate date9.;
run;
```

Partial PROC PRINT Output

SaleDate	Sale Amt	Mth2Dte
01APR2011	498.49	498.49
02APR2011	946.50	1444.99
03APR2011	994.97	2439.96
04APR2011	564.59	3004.55
05APR2011	783.01	3787.56

20

psm04d01

SUM Function

A RETAIN statement along with a SUM function in an assignment statement can be used to create **Mth2Dte**.

```
data mnthtot;
   set orion.aprsales;
   retain Mth2Dte 0;
   Mth2Dte=sum(Mth2Dte,SaleAmt);
run;
```

- The SUM function ignores missing values.

25

Sum Statement

Use the sum statement to create **Mth2Dte**.

```
data mnthtot2;
   set work.aprsales2;
   Mth2Dte+SaleAmt;
run;
```

variable **+** *expression*;

Specifics about **Mth2Dte**:

- initialized to zero
- automatically retained
- increased by the value of **SaleAmt** for each observation
- ignored missing values of **SaleAmt**

26

psm04d02

Sum Statement

```
proc print data=mnthtot2 noobs;
   format SaleDate date9.;
run;
```

Partial PROC PRINT Output (30 Total Observations)

SaleDate	SaleAmt	Mth2Dte
01APR2011	498.49	498.49
02APR2011	.	498.49
03APR2011	994.97	1493.46
04APR2011	564.59	2058.05
05APR2011	783.01	2841.06

27

Accumulator Variables

An *accumulator variable* is a variable that adds on an expression.

Partial Output

Maine County	Population 2000	Total Population	Total Counties
Androscoggin	103,793	103,793	1
Aroostook	73,938	177,731	2
Cumberland	265,612	443,343	3
Franklin	29,467	472,810	4
Hancock	51,791	524,601	5
Kennebec	117,114	641,715	6
Knox	39,618	681,333	7
Lincoln	33,616	714,949	8
Oxford	54,755	769,704	9
Penobscot	144,919	914,623	10

accumulator variables

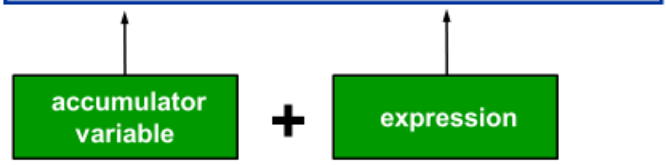
52

Sum Statement

The *sum statement* adds the result of an expression to an accumulator variable.

Examples:

```
TotCounties    + 1;
TotPopulation + Population2000;
```



53

Sum Statement

The accumulator variable has the following characteristics:

- must be a numeric variable
- is automatically set to 0 before SAS reads the first observation
- is retained from one iteration to the next

The expression is defined with the following features:

- is any SAS expression
- is evaluated and the result added to the accumulator variable
- is ignored if missing

54

Sum Statement

The sum statement is equivalent to using the RETAIN statement and the SUM function.

```
TotPopulation + Population2000;
```

```
retain TotPopulation 0;
TotPopulation =
   sum(TotPopulation,Population2000);
```

- The RETAIN statement causes a variable to retain its value from one iteration of the DATA step to the next and specifies an initial value for the variable.
- The SUM function returns the sum of the nonmissing arguments.

55

RETAIN Statement

To initialize an accumulator variable to a value other than zero, include the accumulator variable in a RETAIN statement with an initial value.

	Population	Year
1	914950	1950
2	940841	1955
3	970689	1960
4	993236	1965
5	997357	1970
6	1062640	1975
7	1125027	1980
8	1163850	1985
9	1227928	1990

```
data FiveYearPop;
   set FiveYearPop;
   retain year 1945;
   year+5;
run;
```

58