The George Washington University
Department of Statistics

STAT 6197 - Spring 2019

Week 3 – February 1, 2019

Major Topic: Working with User-Defined Formats and Informats, and Creating New Variables

Detailed Topics:

1. Working with User-Defined Formats
    a. Creating, Applying, Accessing, and Maintaining Formats
    b. Managing Format Catalog Entries and Documenting Formats
    c. Grouping Data Values Using Formats
    d. Removing Formats, Labels, and Informats from SAS Data Sets

2. Working with User-Defined Informats, and Picture Formats

3. Creating New Variables Using
    a. Assignment Statements
    b. SELECT/WHEN/OTHERWISE Statements
    c. Case Expression in PROC SQL
    d. 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

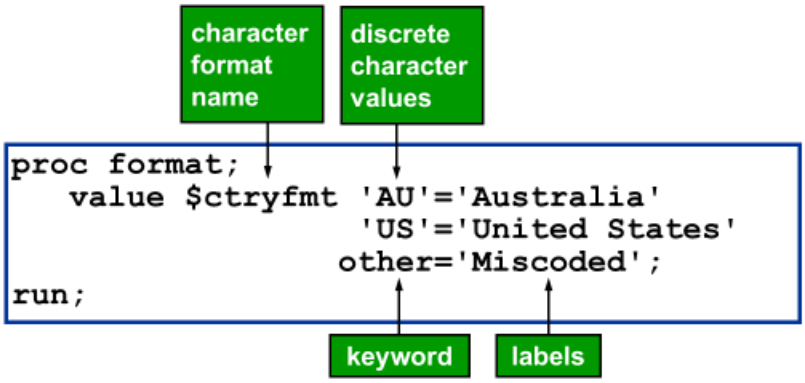
Management has requested that country names, instead of country codes, be used in reports.

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

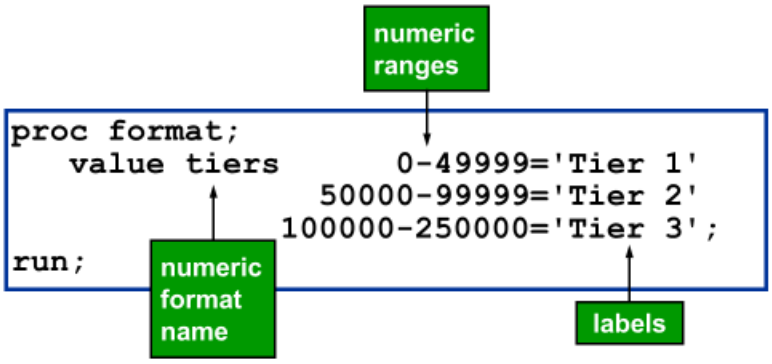
Defining a Character Format



The OTHER keyword includes all values that do not match any other value or range.

26 p105d03

Defining a Numeric Format



37 p105d04

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

GitHub\SASGateway\SASCourse\Week3\Ex1_Numeric_Character_Formats.sas

```
26 proc format;
27    value numfmt
28              Low - <0   = "Low - <0: Nonresponse"
29              0="0: Never"
30              1-5 = "1-5: Within past 5 years"
31              Other = "Other: More than 5 years ago"
32              . ="Missing" ;
33     value $charfmt
34              Low-<'0'   = "Low-<'0': Nonresponse (Missing values included)"
35              '0' = "'0': Never"
36              '1'-'5' = "'1'-'5': Within past 5 years"
37              Other = "Other: More than 5 years ago" ;
38  run;
39 data work.have;
40  input id $ 1 Colonoscopy 3-4 c_Colonoscopy $6-7;
41  datalines;
42  A -1 -1
43  B  .
44  C 3   3
45  D -9 -9
46  F 3  3
47  G 5  5
48  H 6  6
49  I .
50  J 7  7
51  ;
52 proc print data=work.have noobs;
53  Format colonoscopy numfmt. c_colonoscopy $charfmt.;
54  run;
```

id	Colonoscopy	c_Colonoscopy
A	Low - <0: Nonresponse	Low-<'0': Nonresponse (Missing values included)
B	Missing	Low-<'0': Nonresponse (Missing values included)
C	1-5: Within past 5 years	'1'-'5': Within past 5 years
D	Low - <0: Nonresponse	Low-<'0': Nonresponse (Missing values included)
F	1-5: Within past 5 years	'1'-'5': Within past 5 years
G	1-5: Within past 5 years	'1'-'5': Within past 5 years
H	Other: More than 5 years ago	Other: More than 5 years ago
I	Missing	Low-<'0': Nonresponse (Missing values included)
J	Other: More than 5 years ago	Other: More than 5 years ago

Ranges can be specified with special keywords: LOW, HIGH, OTHER, dot(.), blank (' ')

The LOW keyword does not format missing values for numeric formats; it does include missing values for character formats.

The OTHER keyword does include missing values unless accounted for with specification of missing values.

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.

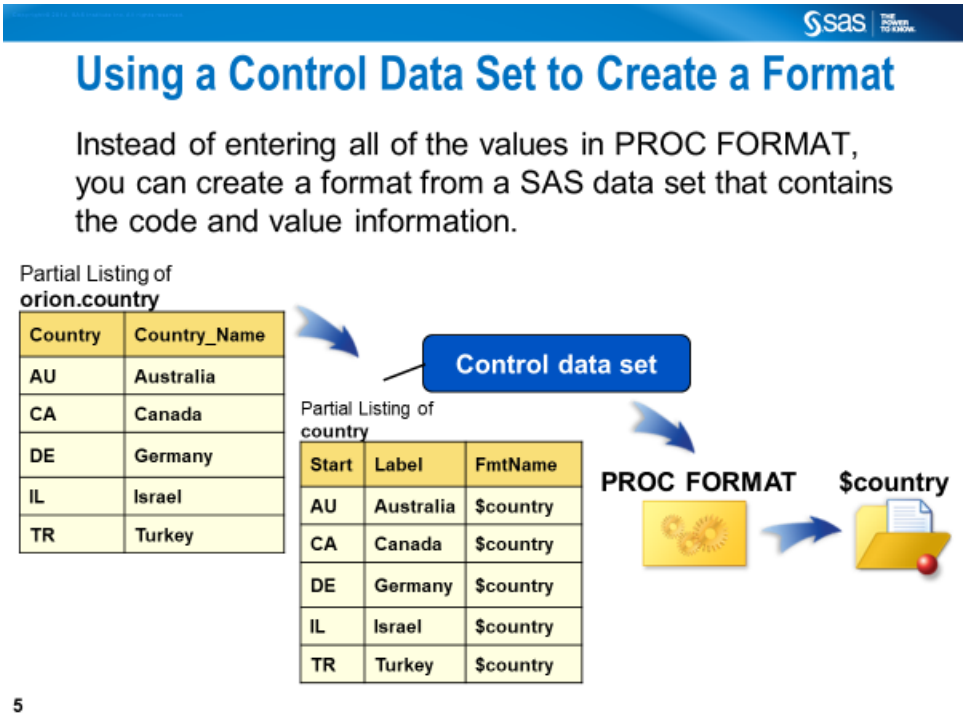
```
2  *Ex2_Nested_Formats.sas;
3  proc format;
4  value date_grp_fmt
5    low-'03jul1995'd = 'Pre July 4th 1995'
6    '04jul1995'd-'31jul1995'd = [mmddyy8.]
7    '01aug1995'd-high = 'Aug 1-Dec 31, 1995';
8  value sales_fmt
9    low-<5000 = 'Less than $5,000'
10   5000-9999 = '$5,000-<$10,000'
11   10000-high = [dollar12.2];
12   run;
13   proc freq data=sashelp.mdv;
14   tables shipdate sales93;
15   format shipdate date_grp_fmt.
16     sales93 sales_fmt.;
17   run;
```

The FREQ Procedure

SHIPDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Pre July 4th 1995	66	51.56	66	51.56
07/09/95	2	1.56	68	53.13
07/11/95	1	0.78	69	53.91
07/17/95	1	0.78	70	54.69
07/18/95	2	1.56	72	56.25
07/23/95	1	0.78	73	57.03
07/26/95	1	0.78	74	57.81
07/29/95	1	0.78	75	58.59
Aug 1-Dec 31, 1995	53	41.41	128	100.00

SALES93	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Less than $5,000	123	96.09	123	96.09
$5,000-<$10,000	3	2.34	126	98.44
$12,063.00	1	0.78	127	99.22
$15,611.00	1	0.78	128	100.00

SALES94	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Less than $5,000	122	95.31	122	95.31
$5,000-<$10,000	3	2.34	125	97.66
$12,245.00	1	0.78	126	98.44
$12,346.00	1	0.78	127	99.22
$15,950.00	1	0.78	128	100.00



Using a Control Data Set to Create a Format

Partial **orion.country**

Country	Country_ Name	Population	Country_ID	Continent_ID	Country_ FormerName
AU	Australia	20,000,000	160	96	
CA	Canada	.	260	91	
DE	Germany	80,000,000	394	93	East/West Germany
IL	Israel	5,000,000	475	95	

DATA Step

Start Label FmtName

```
data country;
   keep Start Label FmtName;
   retain FmtName '$country';
   set orion.country(rename=(Country=Start
                             Country_Name=Label));
run;
```

6

Using a Control Data Set to Create a Format

Use the CNTLIN= option to read the data and create the format.

```
proc format cntlin=country;
run;
```

CNTLIN=*SAS-data-set*

The variables **FmtName**, **Start**, and **Label** are required in order to create a format from a CNTLIN data set.

7

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

Creating a User-Defined Format from a SAS Data Set

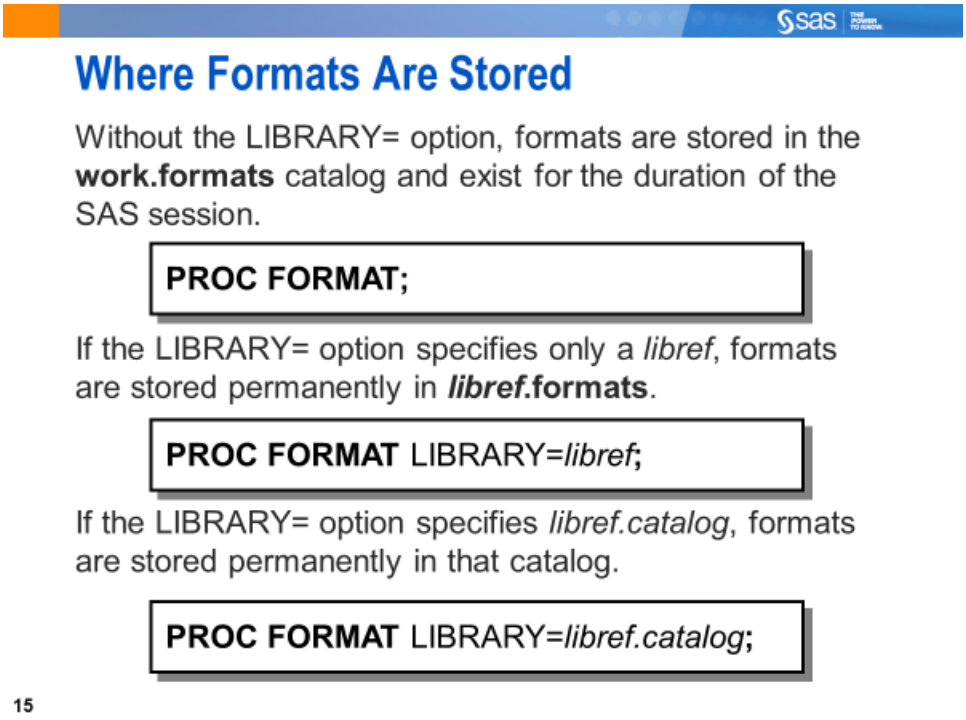
If there is a long list of variable values and if the values and their labels are available in an electronic file (ASCII, EXCEL or data base mode), the file can be read into SAS to create a SAS data set. There is no need to type this long list!

Requirements: The data set must have **three required columns**– FMTNAME, START, and LABEL. The data set can have the optional column called the TYPE column with values of 'C' for the character variable and 'N' for the numeric variable.

The CNTLIN=input-control-SAS-data-set (as shown in line 42 below) specifies a SAS data set from which PROC FORMAT builds INFORMATs. Note that CNTLIN= builds FORMATS and INFORMATS without using a VALUE, PICTURE, or INVALUE statement.

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

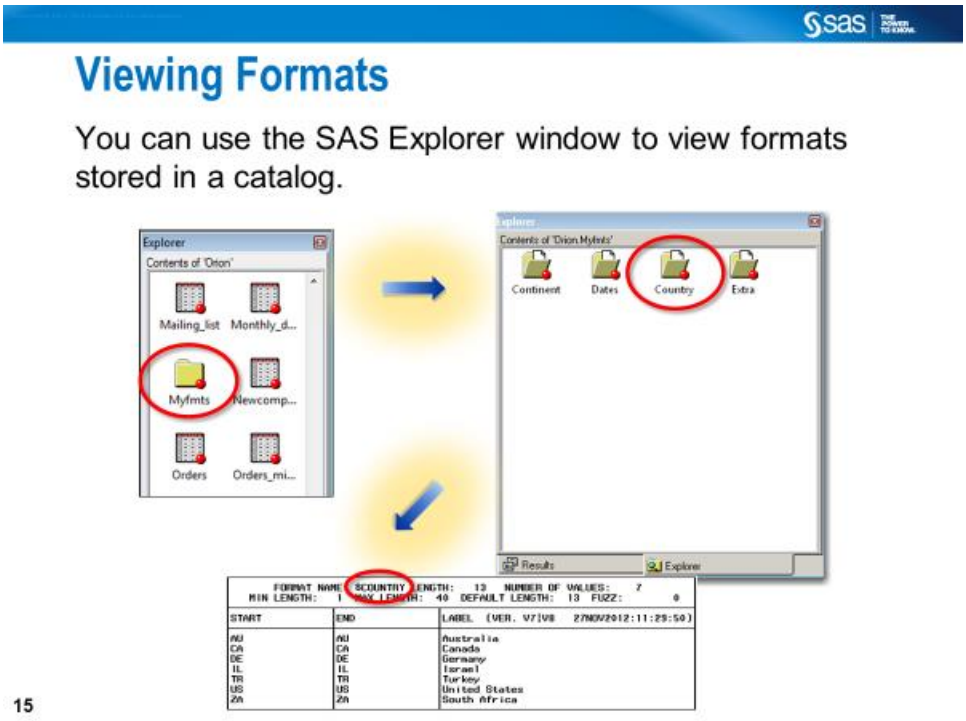
If the LIBRARY= option specifies *libref.catalog*, formats are stored permanently in that catalog.

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

PROC FORMAT LIBRARY=*libref.catalog*;

- Store frequently used formats in permanent catalogs.

14



GitHub\SASGateway\SASCourse\Ex17_Temporary_Permanet_Catalogs.sas

```
1  *Ex17_Temporary_Permanent_Catalogs.sas;
2  PROC FORMAT;
3       value regionfmt
4          1='Northeast' 2='Midwest'
5          3='South'  4='West';
6  run;
```

Creating a User-Defined Format (stored in a permanent catalog)

```
7  LIBNAME library 'C:\SASCourse\Week3';
8  PROC FORMAT LIBRARY=library;
9       value regionfmt
10         1='Northeast' 2='Midwest'
11         3='South'  4='West';
12 run;
```

Lines 7-12: With the LIBRARY=library option specified, the format REGIONFMT is **permanently** stored in a catalog called FORMATS in the folder referenced by the libref **library**.

The LIBRARY= option enables you to specify a SAS library where the formats that you are creating in the PROC FORMAT step are stored.

Creating a User-Defined Format (stored in a permanent catalog)

```
13 LIBNAME sds 'C:\SASCourse\Week3';
14 PROC FORMAT LIBRARY=sds;
15      value x_regionfmt
16        1='Northeast' 2='Midwest'
17        3='South' 4='West';
18 run;
```

Lines 13-18: With the LIBRARY=sds option specified, the format X_REGIONFMT is **permanently** stored in a catalog called FORMATS (named by default) in the folder referenced by the libref **sds. Note that the format and libref are named differently in the above example.**

Creating a User-Defined Format (stored in a permanent catalog that is named by the User)

```
19 LIBNAME xsds 'C:\SASCourse\Week3';
20 PROC FORMAT LIBRARY=xsds.catalogpop;
21       value regionfmt
22          1='Northeast' 2='Midwest'
23          3='South'  4='West';
24 run;
```

Lines 19-24: With the LIBRARY=XSDS.CATALOGPOP specified, formats are **permanently** stored in the catalog called CATALOGPOP (rather than the default folder name FORMATS) in the folder referenced by the libref XSDS.

How to Use Formats (Created and Stored Permanently Earlier)

```
1  *Ex23_read_data_add_format.sas ;
2  OPTIONS nodate nonumber;
3  %LET Path=C:\SASCourse\Week3;
4  FILENAME indata "&Path\pop2013_no_headers.txt";
5  LIBNAME library "&path";
6  LIBNAME SDS "&Path";
7  DATA sds.pop2013x;
8      INFILE indata DLM=',';
9      input sumlev region division fips st_name :$22.
10           pop :comma12. pop18p :comma12.
11           percent_pop18p;
12     FORMAT pop  pop18p  comma12. FIPS z2.
13            region regionfmt. percent_pop18p;
14     LABEL region='Region'
15           FIPS='State*FIPS'
16           st_name ='State*Name'
17           pop='Population*(All Ages)'
18           pop18p='Population*(Aged 18+)'
19           percent_pop18p = 'Percent*Population*(Aged 18+)';
20 run;
21 proc print data=sds.pop2013x noobs split='*';
22   var Region FIPS st_name pop pop18p percent_pop18p;
23   where st_name like 'A%';
24 run;
```

Line 5: Note the LIBNAME statement that associates a libref (named library) with a SAS library (storage location that has format catalog named FORMATS. For users' convenience**,** SAS has included a library called LIBRARY in the search path. So there is no need to use the OPTIONS FMTSEARCH= statement for searching the format catalog FORMATS**.**

See Cody, R. Cody's Data Cleaning Techniques Using SAS® - pp. 16-17 on "Creating Permanent Formats" for further details.

When to Use the FMTSEARCH= System Option

The format catalog is permanently stored in a folder that is referenced by a libref other than **library** (**sds** is the libref in this example) in a LIBNAME statement.

```
1  *Ex19_options_FMTSEARCH.sas;
2  OPTIONS nodate nonumber nocenter;
3  %LET path=C:\SASCourse\Week3;
4  LIBNAME sds "&path";
5  Options FMTSEARCH = (sds.popcatalog);
6  PROC FREQ data=sds.pop2013x;
7   TABLES REGION;
8  RUN;
```

Line 5: Note the FMTSEARCH = options (required) in the OPTIONS statement that tells SAS to search the format catalog **popcatalog** in the folder referenced by the libref **sds**. This catalog contains the format (i.e., REGIONFMT) for the REGION variable that was earlier saved to the SAS library referenced by the libref **sds** and then applied to the same variable in the DATA step when the SAS data set **pop2013x** was created (see Ex23_read_data_add_format).

The FREQ Procedure

Region

region	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Northeast	9	17.65	9	17.65
Midwest	12	23.53	21	41.18
South	17	33.33	38	74.51
West	13	25.49	51	100.00

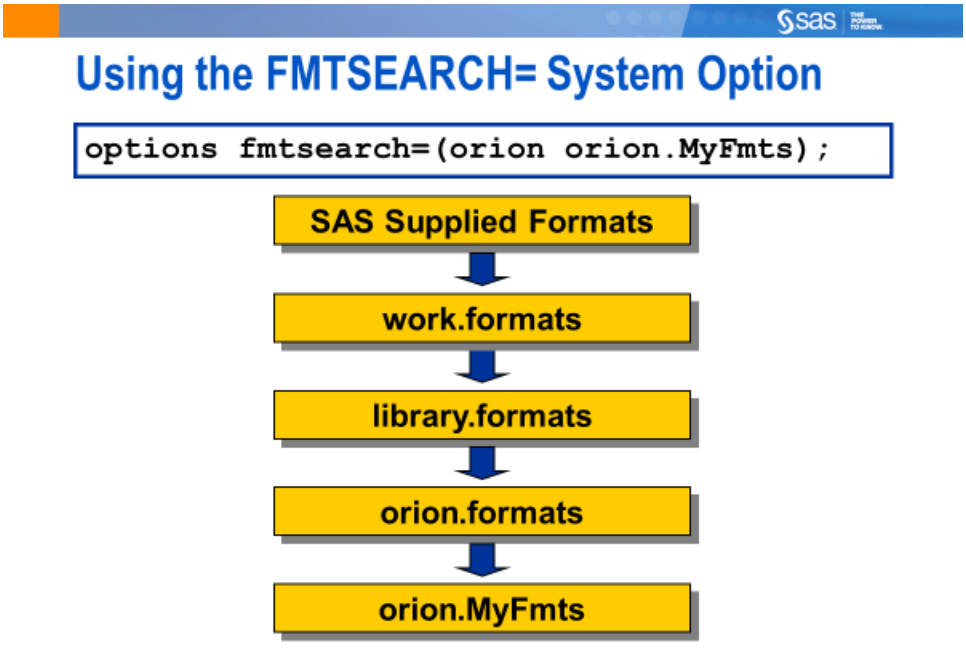
Important Note: Use the NOFMTERR system options to avoid an error message from the SAS system when it cannot find a format for a particular variable.

Using the Library as Libref

Consider the following alternative SAS code fragment that does not use the FMTSEARCH= System Option to search for formats.

```
1  *Ex18_LIBRARY_library.sas;
2  OPTIONS nodate nonumber nocenter;
3  %LET path=C:\SASCourse\Week3;
4  LIBNAME sds "&path";
5  LIBNAME library "&path";;
6  PROC FREQ data=sds.pop2013x;
7   TABLES REGION;
8  RUN;
```

Line 5: Note that SAS has included a library called LIBRARY in the search path. So you don't have to include an OPTIONS FMTSEARCH= statement for searching the catalog FORMATS**.** The FORMATS catalog containing the format (i.e., REGIONFMT) for the REGION variable was earlier saved in C:\SASCourse\Week3.



CATALOG Procedure

The CATALOG procedure manages entries in SAS catalogs.

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

```
PROC CATALOG CATALOG=ilbref.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

How to List the Member(s) of a Format Catalog

PROC CATALOG is used to list the members of a format catalog (e.g., SDS.pop2013catalog as shown below). This catalog includes only one member.

```
2  *Ex5_print_catalog_fmtlib.sas;
3  libname sds 'C:\SASCourse\Week3';
4  proc catalog catalog = sds.catalogpop;
5  contents;
6  run;
```

Formats for Pop2013x Data Set

Contents of Catalog SDS.CATALOGPOP

#	Name	Type	Create Date	Modified Date	Description
1	REGIONFMT	FORMAT	09/10/2017 09:40:43	09/10/2017 09:40:43	

How to Display the Contents of a User-Defined Format

The FMTLIB option of PROC FORMAT displays the start and end values of the format range as well as the label.

```
8  proc format library = sds.catalogpop fmtlib;
9  select REGIONFMT;
10 title 'Formats for Pop2013x Data Set';
11 run;
```

Formats for Pop2013x Data Set

FORMAT NAME: REGIONFMT LENGTH: 9
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 9 FUZZ: STD

START	END	LABEL (VER. 9.4 10SEP2017:09:40:43)
1	1	Northeast
2	2	Midwest
3	3	South
4	4	West

Using Formats to Check for Invalid Data Values

```
1  *Ex6_Finding_Invalids.sas;
2  *Method 1;
3  options nodate nonumber;
4  PROC FORMAT;
5  VALUE date_fmt
6     LOW-HIGH = 'valid date'
7      other='invalid date';
8  run;
9  DATA work.HAVE;
10 infile datalines firstobs=2;
11 input Name $ 1-7
12       @8 s_date ?? mmddyy10.
13       @8 s_date_ch $10.;
14 format s_date mmddyy10. ;
15 datalines;
16 12345678901234567890
17 Alfred 04/22/2005
18 Alice  01/15/2005
19 Barbara12/20/2004
20 Carol  10/29/1999
21 Henry  02/31/2007
22 Philip 02/31/2005
23 Ronald 02/29/2006
24 ;
```

Table lookup using a user-defined format

```
32 proc print DATA=HAVE noobs;
33    var Name s_date_ch;
34    where S_date= .;
35 RUN;
```

The FREQ Procedure

s_date	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Invalid Date	3	42.86	3	42.86
Valid Date	4	57.14	7	100.00

Listing of S_DATE_CH values if the S_DATE has a missing value

```
32 proc print DATA=HAVE noobs;
33    var Name s_date_ch;
34    where S_date= .;
35 RUN;
```

Name	s_date_ch
Henry	02/31/2007
Philip	02/31/2005
Ronald	02/29/2006

See Cody's book - pages 14-16 on "Using Formats to Check for Invalid Values".

How to Delete Labels, Formats, and Informats from SAS Data Sets

GitHub\SASGateway\SASCourse\Week3
Ex24_remove_labels_formats_informats.sas

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.

Picture Statement in the FORMAT Procedure

GitHub\Gateway\SASCourse\Week3\Ex3_picture_statement.sas

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

```
1  *Ex3_Picture_Statement.sas;
2  proc format;
3    picture test (round)
4            low-<0='09.99' (prefix='-')
5            0-<10 ='09.99'
6            10-<100='99.9'
7            100-999 ='999';
8  run;
9  DATA temp;
10 INPUT  Some_value @@;
11  final_value=put(Some_value, test.);
12  datalines;
13   5.209 0.203 -0.027 95.307 357.897
14   ;
15 proc print noobs;  run;
```

Some_ value	final_ value
5.209	5.21
0.203	0.20
-0.027	-0.03
95.307	95.3
357.897	358

```
 97 *Ex3_picture_statement.sas;
 98 PROC FORMAT;
 99  PICTURE p_fmt (ROUND)
100      LOW-<0 = "009.99%" (PREFIX="-" MULT=10000)
101          0-HIGH = "009.99%" (MULT=10000);
102 PICTURE p_fmt_x (ROUND)
103      LOW-<0 = "009.99" (PREFIX="-" MULT=10000)
104          0-HIGH = "009.99" (MULT=10000);
105 RUN;
106 DATA work.have;
107  INPUT Value1 @@;
108  Value2 = Value1; Value3 = Value1;
109  Value4 = Value1; Value5 = Value1;
110 DATALINES4;
111 0.0345678  -0.00123456  -0.456789 .120
112 ;;;;
113 options nodate nonumber;
114 TITLE 'SAS- and User-Defined Formats Applied';
115 PROC PRINT DATA=work.have SPLIT="*" NOOBS;
116  VAR Value: ;
117  FORMAT Value2 PERCENT8.2  Value3 PERCENTN8.2
118         Value4 p_fmt. Value5 p_fmt_x.;
119  LABEL Value1="No"*"Format"*"Applied"
120        Value2="SAS"*"Percent"*"Format"*"PERCENT8.2"*"Applied"
121        Value3="SAS"*"Percent"*"Format"*"PERCENTN8.2"*"Applied"
122        Value4="User"*"Picture"*"Format 1"*"Applied"
123        Value5="User"*"Picture"*"Format 2"*"Applied";
124 RUN;
```

SAS- and User-Defined Formats Applied

No Format Applied	SAS Percent Format PERCENT8.2 Applied	SAS Percent Format PERCENTN8.2 Applied	User Picture Format 1 Applied	User Picture Format 2 Applied
0.03457	3.46%	3.46%	3.46%	3.46
-0.00123	(0.12%)	-0.12%	-0.12%	-0.12
-0.45679	(45.68%)	-45.68%	-45.68%	-45.68
0.12000	12.00%	12.00%	12.00%	12.00

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.

Working with User Defined Informats: Invalue Statement in the FORMAT Procedure

This statement reads and converts the raw data values using the INVALUE statement. Here we use the INVALUE statement to create an INFORMAT to convert the character string into a numeric variable while reading the data into SAS.

```
1  *Ex8_Invalue_statement.sas;
2  *** Part 1;
3  proc format ;
4      invalue scorefmt (upcase just)
5              'A'=95  'B'=84
6              'C'=79  'D'=60;
7  run;
8  data Grade_data1;
9    input @1 id $4.  @6 grade scorefmt2. ;
10 datalines;
11 S001 A
12 S002 D
13 S003 B
14 S004 B
15 S005  D
16 S006 C
17 S007 c
18 ;
19 title 'Invalue Statement with UPCASE and JUST Options';
20 proc print data=Grade_data1 noobs ; run;
```

Lines 5-6: The values to the right-hand side of the equal sign are not in quotes.
Line 9: scorefmt2. is a numeric informat, which creates a numeric variable GRADE.

Invalue Statement with UPCASE and JUST Options

id	grade
S001	95
S002	60
S003	84
S004	84
S005	60
S006	79
S007	79

Invalue Statement in the FORMAT Procedure (continued – additional example)

```
22 *** Part 2;
23 proc format ;
24      invalue gpafmt (upcase)
25              2.5-4.0 = _SAME_
26              'B' = 3.0
27              OTHER=.
28 ;
29 data GPA_data;
30   input id $  GPA :gpafmt3. @@;
31 datalines;
32 S001 2.8 S002 3.7 S003 4.0 S004 B
33 S005 2.7 S006 3.2 S007 .
34 ;
35 title 'Invalue Statement with the UPCASE Option';
36 title2 'and _SAME_ and OTHER Keywords';
37 proc print data=GPA_data noobs ; run;
```

```
Invalue Statement with the UPCASE Option
     and _SAME_ and OTHER Keywords

              id     GPA

            S001     2.8
            S002     3.7
            S003     4.0
            S004     3.0
            S005     2.7
            S006     3.2
            S007      .
```

Lines 25-26: The values to the right-hand side of the equal sign are not in quotes.
Line 30: gpafmt3. is a numeric informat, which creates a numeric variable GPA.

See GitHUB/SASGateway/SASCourse/WEEK3/Ex8_Invalue_Statement for more examples

Creating New Variables using the Assignment Statement

Completed Program

Place the assignment statement in the DATA step.

```
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;
run;
```

49 in09d01

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.

Ex9_Create_vars_Different_Ways.sas

1. **Creating New Variables Using the Assignment Statement**
2. **Creating Lookup Values (i.e., New Variables) Using IF-THEN-ELSE Statements – Grouping Values**
3. **Table Lookup Using PROC FORMAT**

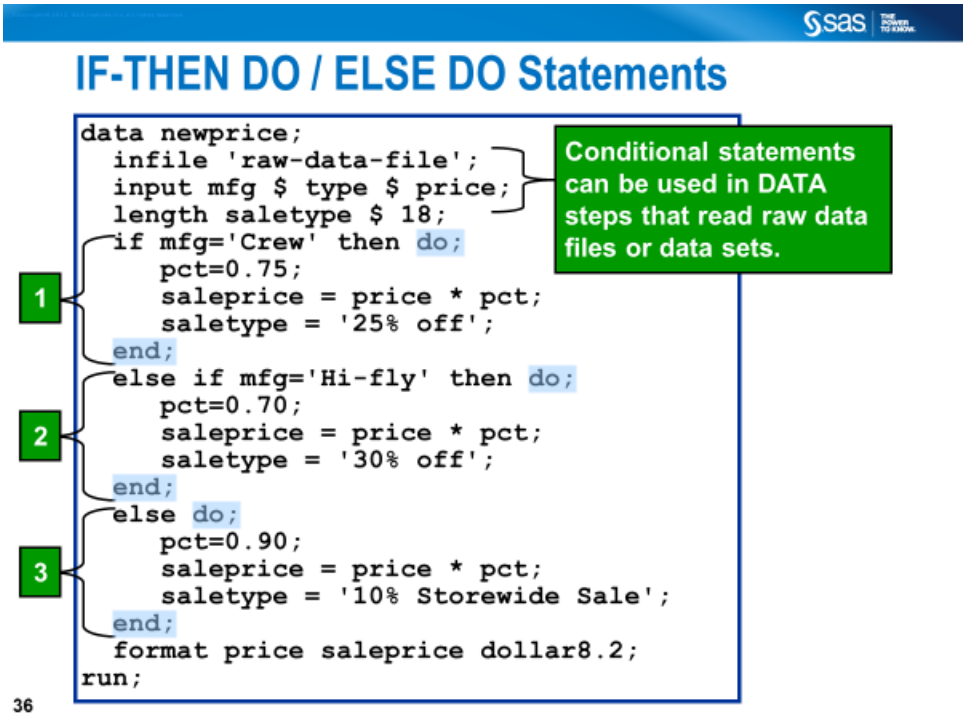
```
49 /* Many-to-1 mappings of values
50 into literal labels */
51 PROC FORMAT;
52   VALUE $pf_fmt
53       'A'-<'D' = 'PASS'
54       OTHER    = 'FAILED';
55 PROC FREQ DATA=new_class;
56  table Grade /missing;
57  format Grade $pf_fmt.;
58 RUN;
```

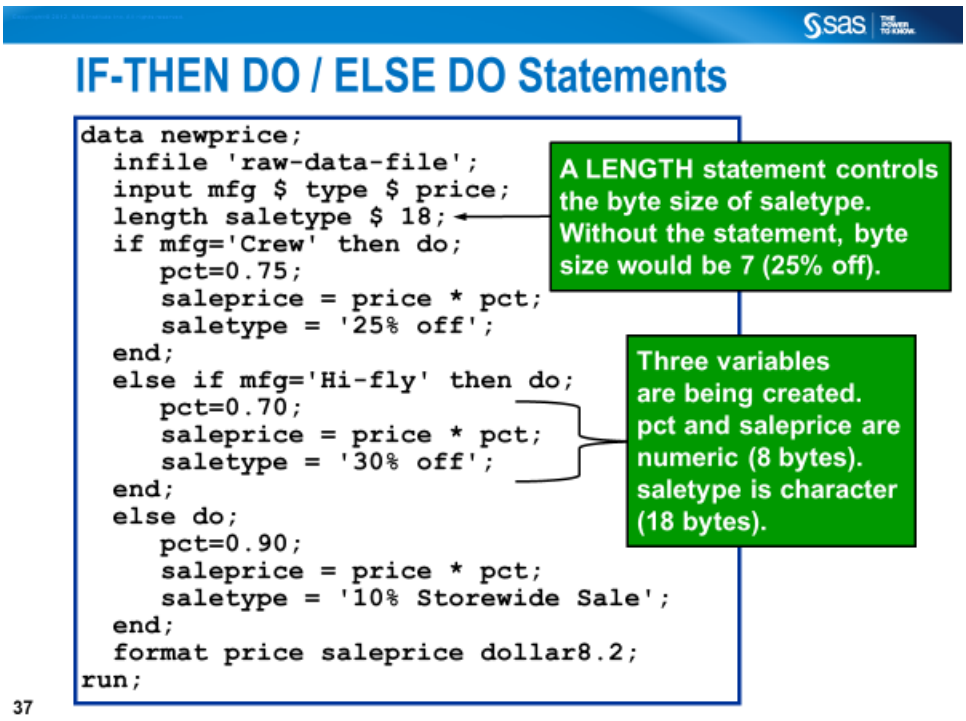
Creating a user-defined format can be viewed as a table look-up that uses 1-to-1 or many-to-1 mappings of values.

A temporary format is applied to the variable GRADE in PROC FREQ step.

This is an alternative to the IF-THEN-ELSE code in data step.

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





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.

```
1  *Ex10_DO_Group.sas;
2  data age_data;
3   length age_group $14 xage_group $25;
4   input age @@ ;
5    if age <=17 then
6        DO;
7          age_group ='<=17 Years';
8          xage_group= 'Infants-Young Adolesents';
9        END;
10  else if 18<=age<=34 then
11       DO;
12         age_group= '18-34 Years';
13         xage_group= 'Young Adults';
14       END;
15
16  else if 35<=age<=54 then
17       DO;
18         age_group= '35-54 Years';
19         xage_group='Middle Aged Adults';
20       END;
21  else if age>=55 then
22      DO;
23        age_group= '55+ Years';
24        xage_group= 'Older Adults';
25      END;
26   datalines;
27  0 5 10 17 40 48 50 59 62 81 99 100
28  ;
29 proc print noobs; var age age_group xage_group; run;
```

age	age_group	xage_group
0	<=17 Years	Infants-Young Adolesents
5	<=17 Years	Infants-Young Adolesents
10	<=17 Years	Infants-Young Adolesents
17	<=17 Years	Infants-Young Adolesents
40	35-54 Years	Middle Aged Adults
48	35-54 Years	Middle Aged Adults
50	35-54 Years	Middle Aged Adults
59	55+ Years	Older Adults
62	55+ Years	Older Adults
81	55+ Years	Older Adults
99	55+ Years	Older Adults
100	55+ Years	Older Adults

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 Lookup Values (i.e., New Variables) Using SELECT Group in a DATA Step

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.

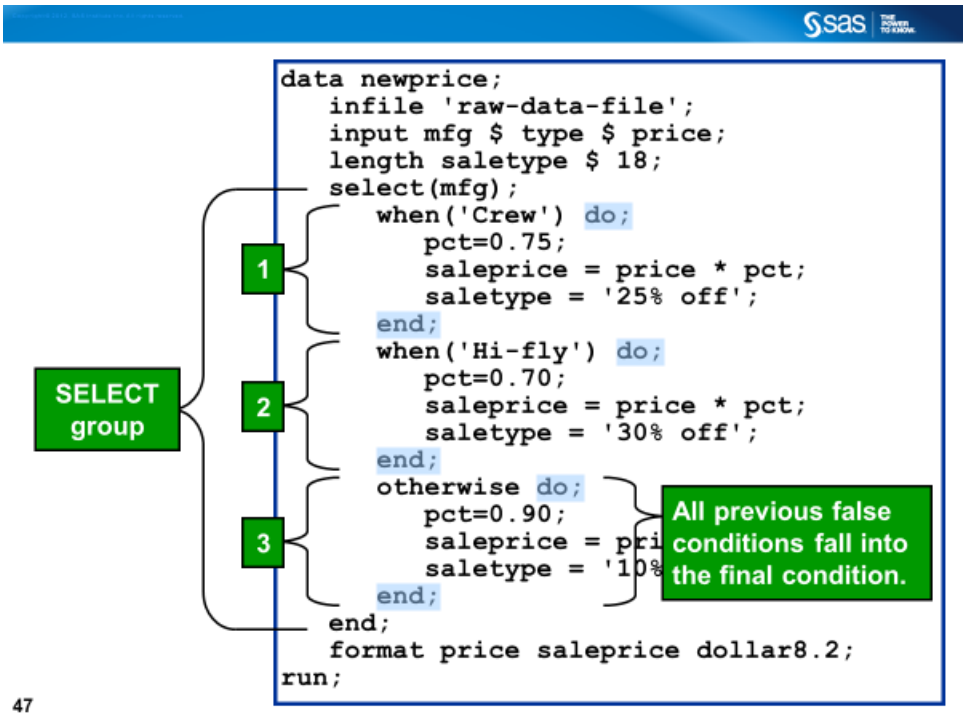
```
ERROR: Unsatisfied WHEN clause and no
OTHERWISE clause at line 618 column 3.
mfg=X-treme type=Strata price=$10.60
saleprice=. _ERROR_=1 _N_=10
```

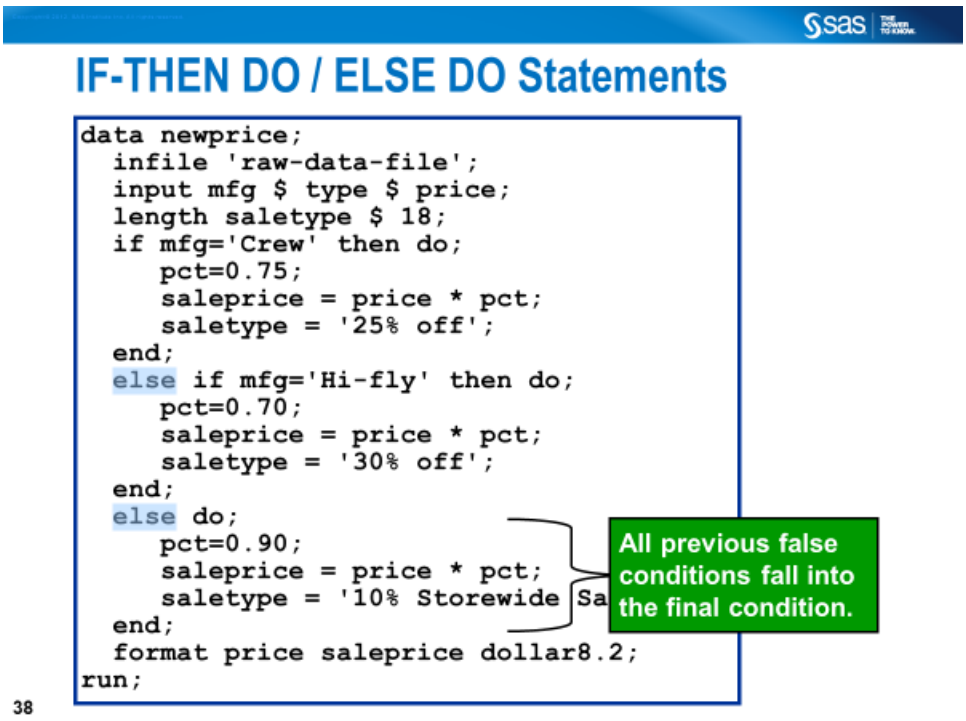
```
select(mfg);
   when('Crew') saleprice=price*0.75;
   when('Hi-fly') saleprice=price*0.70;
   when('White') saleprice=price*0.90;
end;
```

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

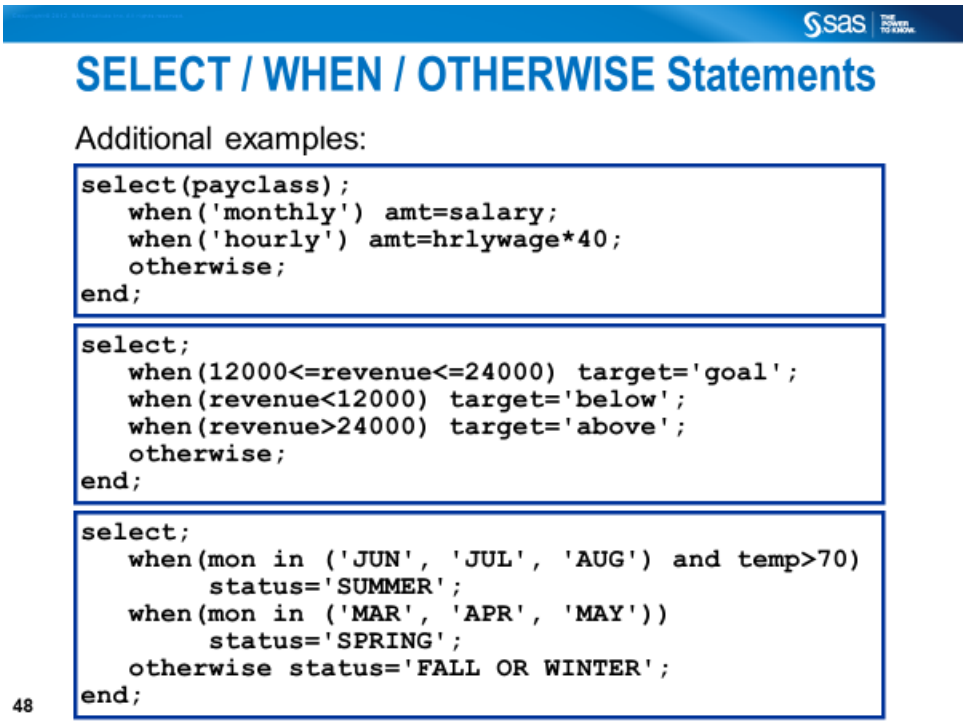
no ERROR

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.



```
1  *Ex20_SELECT_WHEN.sas;
2  data month_data;
3     LENGTH Season $ 12 Quarter $ 9;
4    INPUT Month $ @@;
5      If Month in ('Jun', 'Jul', 'Aug') then Season = 'Summer';
6      else if Month in ('Mar', 'Apr', 'May') then Season ='Spring';
7      else Season = 'Fall/Winter';
8
9    select (Month);
10     when ('Jan','Feb','Mar')   Quarter = 'Quarter 1';
11     when ('Apr', 'May', 'Jun') Quarter = 'Quarter 2';
12     when ('Jul', 'Aug', 'Sep') Quarter = 'Quarter 3';
13     when ('Oct', 'Nov', 'Dec') Quarter = 'Quarter 4';
14     otherwise;
15   end;
16    datalines;
17  Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
18  ;
19  run;
20 proc print noobs; var Month Season Quarter ; run;
```

SELECT groups perform conditional processing

- start with the SELECT statement (optional SAS expression)
- end with the END statement
- contain at least 1 WHEN statement (executed when a WHEN-condition is met
- may contain an OTHERWISE statement (executed if none of the WHEN-conditions is met)

[Point to remember: SAS issues an error message when the result of all SELECT-WHEN statements is false and no OTHERWISE statement is present.]

Month	Season	Quarter
Jan	Fall/Winter	Quarter 1
Feb	Fall/Winter	Quarter 1
Mar	Spring	Quarter 1
Apr	Spring	Quarter 2
May	Spring	Quarter 2
Jun	Summer	Quarter 2
Jul	Summer	Quarter 3
Aug	Summer	Quarter 3
Sep	Fall/Winter	Quarter 3
Oct	Fall/Winter	Quarter 4
Nov	Fall/Winter	Quarter 4
Dec	Fall/Winter	Quarter 4

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;
```

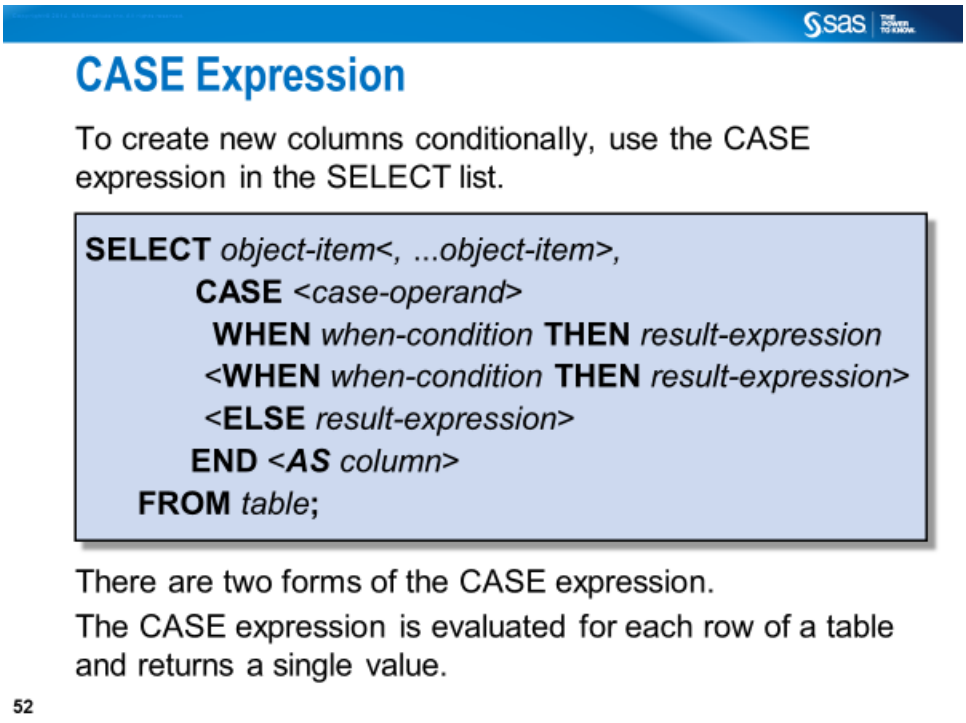
The specified order of the above clauses within the SELECT statement is required.

12

Some **F**rench **W**omen **G**ive **H**appy **O**rders.

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 New Variables Using Case Expression in PROC SQL



```
1  *Ex21_create_variable_in_SQL.sas;
2  options nodate nonumber;
3  PROC SQL;
4  SELECT name, weight format=6.1,
5    CASE
6    WHEN weight <100 THEN '<100 lbs'
7    WHEN weight GE 100 AND weight LT 120 THEN '100-<120 lbs'
8    WHEN weight GE 120 AND weight LE 150 THEN '120-150 lbs'
9    ELSE '120-150 lbs'
10   END AS Weight_Cat label= 'Weight Category'
11 FROM sashelp.class
12 ORDER BY weight;
13 QUIT;
```

Name	Weight	Weight Category
Joyce	50.5	<100 lbs
Louise	77.0	<100 lbs
James	83.0	<100 lbs
Alice	84.0	<100 lbs
Jeffrey	84.0	<100 lbs
Jane	84.5	<100 lbs
Thomas	85.0	<100 lbs
Judy	90.0	<100 lbs
Barbara	98.0	<100 lbs
John	99.5	<100 lbs
Henry	102.5	100-<120 lbs
Carol	102.5	100-<120 lbs
Mary	112.0	100-<120 lbs
William	112.0	100-<120 lbs
Alfred	112.5	100-<120 lbs
Janet	112.5	100-<120 lbs
Robert	128.0	120-150 lbs
Ronald	133.0	120-150 lbs
Philip	150.0	120-150 lbs

Creating a New Variable with the IFC Function

The IFC function normally uses three arguments

- 1st argument - a logical expression () – a condition (true/false) to be evaluated
- 2nd argument - character value returned when true
- 3rd argument – character value to returned when false

```
1  *Ex12_IFC_IFN_Function.sas;
2  proc format;
3      value agefmt
4          . = 'Unknown'
5          1 = 'TRUE'
6          0 = 'FALSE'
7  ;
8
9  data IFC1_IFN1;
10   length age 3 age_group_IFC1 $10;
11   input age @@ ;
12     age_group_IFC1 = IFC(0<=age<=64, '0-64 Years', '65+ Years');
13     age_LE64_IFN1 = IFN(0<=age<=64, 1, 0);
14     age_LE64_IFN1_formatted = put(age_LE64_IFN1, agefmt.);
15     drop age_LE64_IFN1;
16   datalines;
17   0 5 10 17 40 48 50 59 62 81 99 100
18   ;
19 proc print noobs; run;
```

age	age_group_IFC1	age_LE64_IFN1_formatted
0	0-64 Years	TRUE
5	0-64 Years	TRUE
10	0-64 Years	TRUE
17	0-64 Years	TRUE
40	0-64 Years	TRUE
48	0-64 Years	TRUE
50	0-64 Years	TRUE
59	0-64 Years	TRUE
62	0-64 Years	TRUE
81	65+ Years	FALSE
99	65+ Years	FALSE
100	65+ Years	FALSE

Creating a New Variable with the IFN Function

It is the same as the IFC function except that the IFN function returns the numeric value. For a logical expression with a missing value as in the following example, you can have a 4th argument for SAS to return the value in the 4th argument.

```
 1 *Ex13_IFN_Fourth_Argument.sas;
 2 DATA Work.Ifn_Func;
 3 INPUT property_value;
 4 property_tax = ifn(property_value GE 150000,
 5                    property_value*.02,
 6                    property_value*.015, .);
 7 format property_value property_tax dollar8.;
 8 datalines;
 9 150000
10 .
11 250000
12 100000
13 ;
14 proc print data=Work.Ifn_Func;
15 run;
```

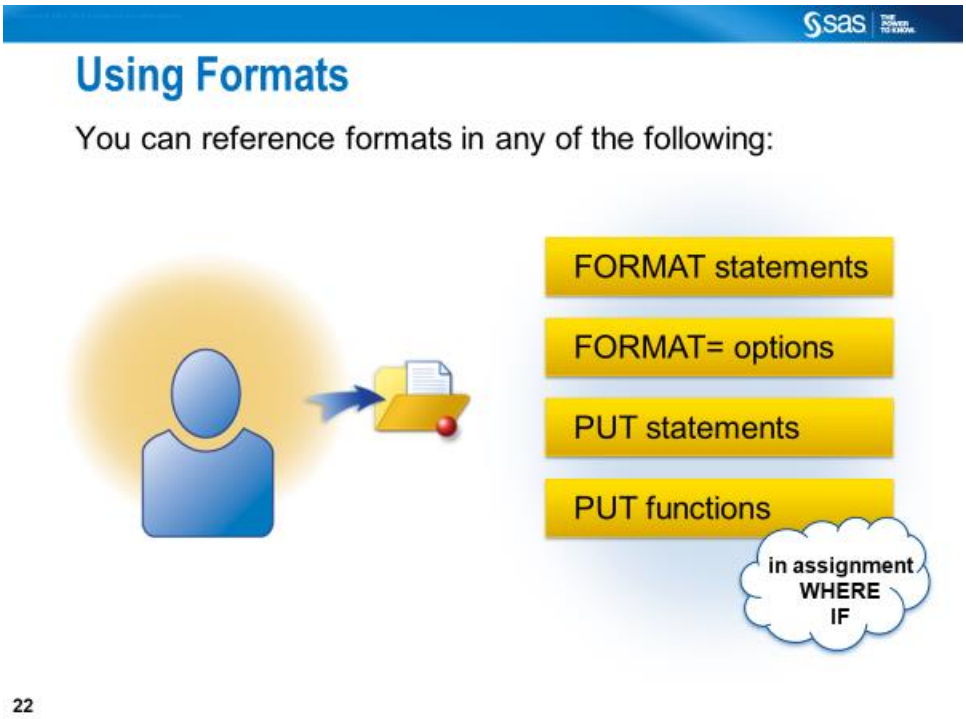
Obs	property_ value	property_ tax
1	$150,000	$3,000
2	.	.
3	$250,000	$5,000
4	$100,000	$1,500

Creating New Variables with WHICHC and WHICHN Functions

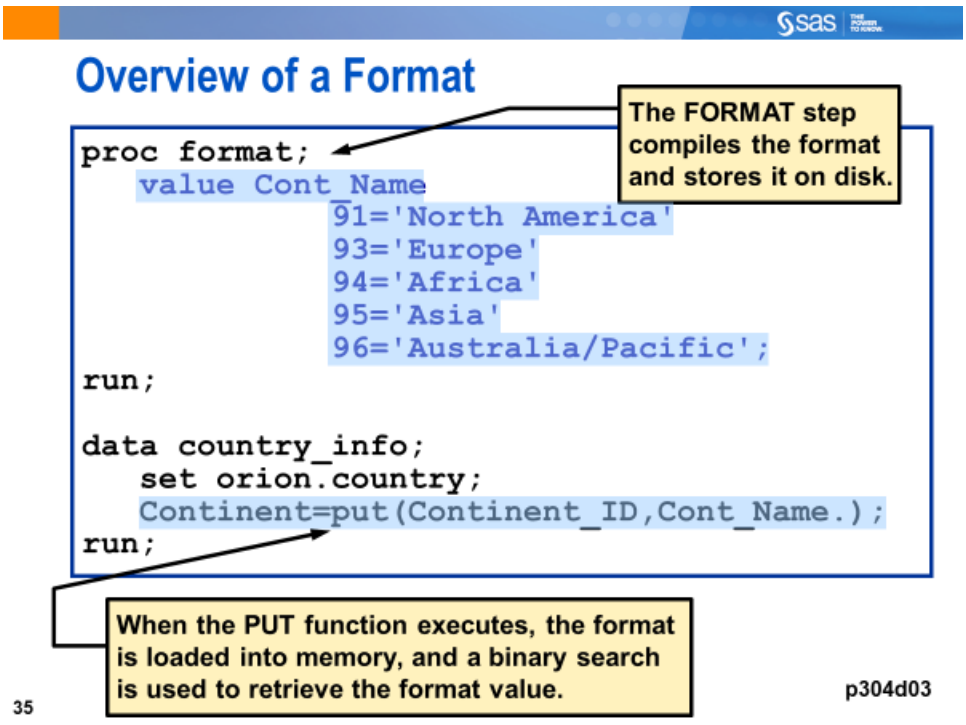
```
1  *Ex14_in_whichn_whichc.sas;
2  * Adapted from Joe Matise's code  (SAS-L) - 5/16/2016;
3  data have1;
4     set sashelp.class (obs=3);
5     Age_13_dummy = whichn(13,age);
6     Gender_male_dummy= whichc('M',sex);
7   run;
8   title ' Listing from HAVE1 Data';
9  proc print data=have1 noobs;  run;
```

Listing from HAVE1 Data

Name	Sex	Age	Height	Weight	Age_13_ dummy	Gender_ male_ dummy
Alfred	M	14	69.0	112.5	0	1
Alice	F	13	56.5	84.0	1	0
Barbara	F	13	65.3	98.0	1	0



Creating Variables Using PUT Function and Formats



Grouping Values Using User-Defined Formats

```
1  *Ex11_Format_Put_Function.sas;
2  options nocenter nonumber nodate;
3  PROC FORMAT;
4       value regionfmt
5          1='Northeast' 2='Midwest'
6          3='South'  4='West';
7  run;
8  LIBNAME SDS "C:\SASCourse\Week3";
9  data Have;
10   set sds.pop2013x;
11   region_char=put(region, regionfmt.);
12 run;
13 proc contents data=Have
14      (keep=region region_char) varnum;
15 ods select position;
16 run;
17 proc print data=Have;
18   var region region_char st_name;
19   where st_name like 'A%';
20 run;
```

```
The CONTENTS Procedure

              Variables in Creation Order

#    Variable       Type    Len    Format        Label

2    region         Num       8    REGIONFMT.    Region
9    region_char    Char      9
```

Obs	region	region_ char	st_name
1	South	South	Alabama
2	West	West	Alaska
3	West	West	Arizona
4	South	South	Arkansas

Creating Character-Type Categorical Variables Using a PUT Function in the Assignment Statement

```
52  *Ex9_Create_vars_Different_Ways.sas (part 4);
53  proc format ;
54  value agefmt low-17 = '0-17 Years'
55               18-49 = '18-49 Years'
56               50-64 = '50-64 Years'
57               65-High = '65+ Years'
58               ;
59   data Have4;
60     input age @@ ;
61     *Character-type categorical variables using a PUT function;
62     age_group_x=put(age, agefmt.);
63   datalines;
64   0 5 10 17 40 48 50 59 62 81 99 100
65   ;
66  title 'Have4 Data';
67  proc freq data=have4;
68     table age_group_x ;
69  run;
```

Have4 Data

The FREQ Procedure

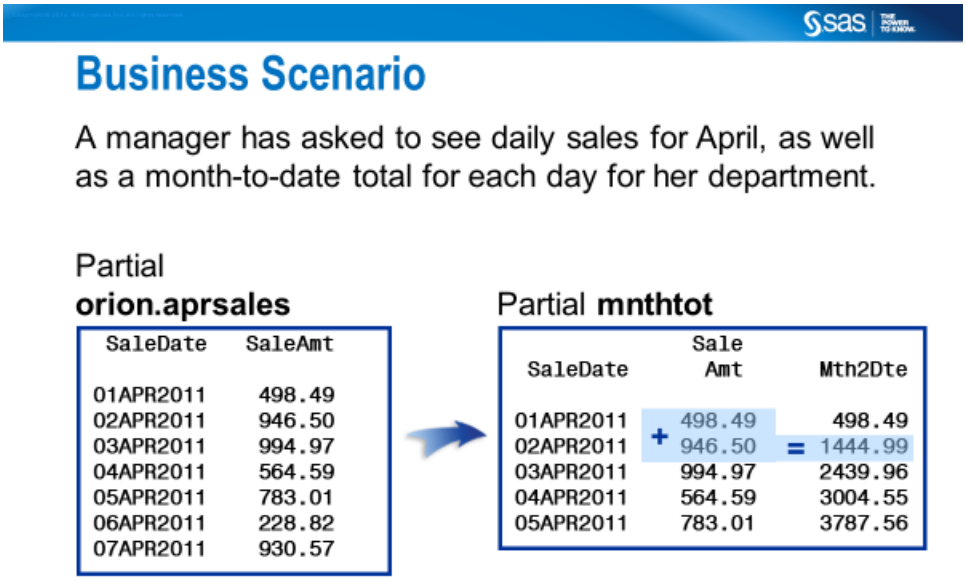
age_group_x	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0-17 Years	4	33.33	4	33.33
18-49 Years	2	16.67	6	50.00
50-64 Years	3	25.00	9	75.00
65+ Years	3	25.00	12	100.00

See Cody, R. Cody's Data Cleaning Techniques Using SAS®; page 33 program 3.3 for further explanation of the SAS code on creating new variables using the PUT function and Format.

See the format that overlapping data values GitHub/SASGateway/SASCourse/Ex25_date_group.sas.

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 a New Variable using the SUM Statement



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

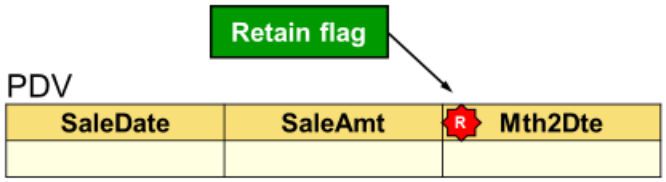
- retains the value of the variable in the PDV across iterations of the DATA step
- initializes the retained variable to missing or a specified initial value before the first iteration of the DATA step
- is a compile-time-only statement.

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

Setup for the Poll

What happens if there are missing values for **SaleAmt**?

Open and submit **psm04a02** and examine the output.

Partial **orion.Aprsales2**

Sale Date	SaleAmt
18718	498.49
18719	.
18720	994.97

Missing value for SaleAmt

21

4.02 Multiple Choice Poll – Correct Answer

What effect did the missing value for **SaleAmt** have on **Mth2Dte**?

a. The missing value is ignored; **Mth2Dte** values are not affected.
b. The missing value causes the DATA step to stop processing.
c. The missing value causes the subsequent values for **Mth2Dte** to be set to missing.

23

Undesirable Output: Missing Values

SaleDate	Sale Amt	Mth2Dte
01APR2011	498.49	498.49
02APR2011	.	.
03APR2011	994.97	.
04APR2011	564.59	.
05APR2011	783.01	.

missing value

Subsequent values of Mth2Dte are missing.

24

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

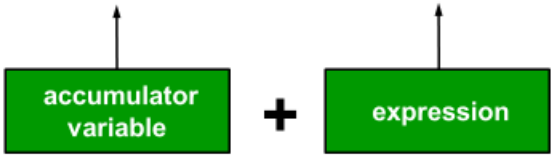
52

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:

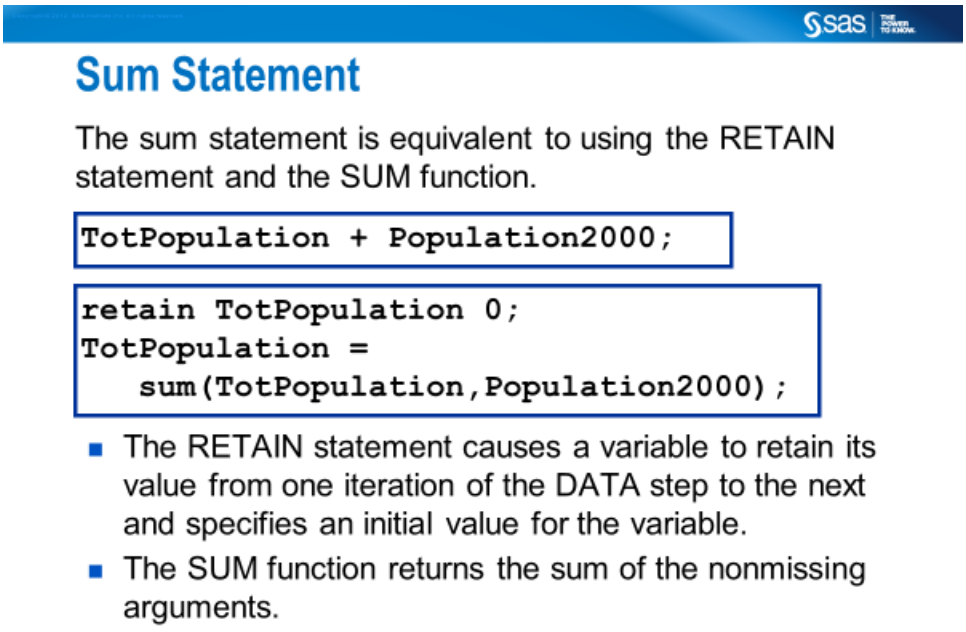
- is any SAS expression
- is evaluated and the result added to the accumulator variable
- is ignored if missing

54

Another Example Program

```
1  *Ex22_Retain_Sum_Statement.sas;
2  DATA temp ;
3    INPUT month sales @@;
4        Total_sales+sales;
5   FORMAT Sales Total_sales dollar8.;
6    DATALINES;
7      1 4000 2 5000 3 . 4 5500 5 5000
8      ;
9  PROC PRINT noobs; run;
```

month	sales	Total_ sales
1	$4,000	$4,000
2	$5,000	$9,000
3	.	$9,000
4	$5,500	$14,500
5	$5,000	$19,500



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

Rules of Creating a Retained Variable Using RETAIN Statement

A retained variable

- is not automatically set to missing before the next iteration of the data step (except for SET, MERGE, and UPDATE)
- value is retained from the current iteration to the beginning of the next iteration of the data step
- any change in value is controlled by the programmer

The RETAIN statement

- names the variables which should **not** be set to missing before the next iteration of the data step (the "retained variables")
- may give initial values (for first iteration of data step)
- non-executable (can be placed anywhere in the data step)
- examples:

  retain x1 0;

  retain x1-x5 0 name 'alice';

  retain x1-x5 (0 1 2 3 4);

```
11  *Ex22_Retain_Sum_Statement.sas;
12  DATA temp1 ;
13     RETAIN Total_sales 0;
14     FORMAT Sales Total_sales dollar8.;
15     INPUT month sales @@;
16      Total_sales= sum(Total_sales, sales);
17     DATALINES;
18     1 4000 2 5000 3 . 4 5500 5 5000
19     ;
20  PROC PRINT data=temp1;
21    VAR month sales Total_sales;
22  run;
```

Obs	month	Sales	Total_ sales
1	1	$4,000	$4,000
2	2	$5,000	$9,000
3	3	.	$9,000
4	4	$5,500	$14,500
5	5	$5,000	$19,500

Overriding the Default Behavior of the Sum Variable

By default, the sum variable is automatically set to 0 before the first observation is read. To reset the sum variable to a different number, you need to use the RETAIN statement. Below is an example.

```
25 *Ex22_Retain_Sum_Statement.sas;
26 DATA temp;
27     RETAIN Total_sales 1000;
28     INPUT month sales @@;
29      Total_sales+sales;
30     FORMAT Sales Total_sales dollar8.;
31     DATALINES;
32     1 4000 2 5000 3 . 4 5500 5 5000
33     ;
34 PROC PRINT noobs;
35  var month sales Total_sales;
36 RUN;
```

month	sales	Total_ sales
1	$4,000	$5,000
2	$5,000	$10,000
3	.	$10,000
4	$5,500	$15,500
5	$5,000	$20,500

Reordering Data Set Variables

The RETAIN statement can be used in a DATA step to permanently change the order of the variables in an existing data set.

```
data annual_orders;
   retain Customer_ID Month1-Month12;
   set annual_orders;
run;
```

The data set **annual_orders** is used for input and output.

It is recommended that no additional processing be performed in the DATA step.

116 p2aad19