STAT604 SAS Lesson 10

Producing Basic Reports

The PRINT Procedure – Prep Guide Chapter 6

PRINT Procedure

By default, PROC PRINT displays all observations, all variables, and an Obs column on the left side.

```
proc print data=cert.therapy;
run;
```

Partial PROC PRINT Output

Obs	Date	AerClass	WalkJogRun	Swim
1	JAN2012	56	78	14
2	FEB2012	32	109	19
3	MAR2012	35	106	22
4	APR2012	47	115	24
5	MAY2012	55	121	31

Statements and options can be added to the PRINT procedure to modify the default behavior.

Viewing the Output

In this output, two lines are used for each observation.

```
                                          Customer_    Customer_
Obs     Customer_ID    Customer_Name        Gender     Country          Customer_Group

 37              79    Najma Hicks            F         US         Orion Club members
 58           11171    Bill Cuddy             M         CA         Orion Club Gold members
 66           46966    Lauren Krasowski       F         CA         Orion Club members
 ...
 76           70210    Alex Santinello        M         CA         Orion Club members

        Customer_
Obs     Age_Group                 Customer_Type

 37    15-30 years    Orion  Club members medium activity
 58    15-30 years    Orion Club Gold members low activity
 66    15-30 years    Orion  Club members high activity
 ...
 76    15-30 years    Orion  Club members medium activity
```

The Obs column helps identify observations that span multiple lines in a report.

VAR Statement

The VAR statement selects variables to include in the report and specifies their order.

```
proc print data=cert.admit;
   var age height weight fee;
run;
```

VAR *variable(s)*;

Partial PROC PRINT Output

Obs	Age	Height	Weight	Fee
1	27	72	168	85.20
2	34	66	152	124.80
3	31	61	123	149.75
4	43	63	137	149.75
5	51	71	158	124.80

Suppressing the Obs Column

Use the NOOBS option in the PROC PRINT statement to suppress the Obs column.

```
proc print data=cert.admit noobs;
   var age height weight fee;
run;
```

PROC PRINT DATA=*SAS-data-set* NOOBS;

PROC PRINT Partial Output

Age	Height	Weight	Fee
27	72	168	85.20
34	66	152	124.80
31	61	123	149.75
43	63	137	149.75
51	71	158	124.80

ID Statement

The *ID statement* specifies the variable or variables to print at the beginning of each row instead of an observation number.

```
proc print data=cert.reps;
   id lastname idnum;
run;
```

ID *variables*;

LastName	IDnum	FirstName	City	State	Sex	JobCode	Salary	Birth	Hired	HomePhone
CASTON	1269	FRANKLIN	STAMFORD	CT	M	NA1	41690.00	06MAY60	01DEC80	203/781-3335
FERNANDEZ	1935	KATRINA	BRIDGEPORT	CT		NA2	51081.00	31MAR42	19OCT69	203/675-2962
NEWKIRK	1417	WILLIAM	PATERSON	NJ	,	NA2	52270.00	30JUN52	10MAR77	201/732-6611
NORRIS	1839	DIANE	NEW YORK	YN	F	NA1	43433.00	02DEC58	06JUL81	718/384-1767

Choose ID variables that uniquely identify observations.

ID Statement

IDnum	LastName	FirstName	City	State	Sex	JobCode	Salary	Birth	Hired
1269	CASTON	FRANKLIN	STAMFORD	CT	M	NA1	41690.00	06MAY60	01DEC80
1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT		NA2	51081.00	31MAR42	19OCT69
1417	NEWKIRK	WILLIAM	PATERSON	NJ	.	NA2	52270.00	30JUN52	10MAR77
1839	NORRIS	DIANE	NEW YORK	YN	F	NA1	43433.00	02DEC58	06JUL81
1111	RHODES	JEREMY	PRINCETON	NJ	M	NA1	40586.00	17JUL61	03NOV80
1352	RIVERS	SIMON	NEW YORK	NY	M	NA2	5379.80	05DEC48	19OCT74
1332	STEPHENSON	ADAM	BRIDGEPORT	CT	M	NA1	42178.00	20SEP58	07JUN79
1443	WELLS	AGNES	STAMFORD	CT	F	NA1	422.74	20NOV56	01SEP79

IDnum	LastName	HomePhone	birth_month	area
1269	CASTON	203/781-3335	5	203
1935	FERNANDEZ	203/675-2962	3	203
1417	NEWKIRK	201/732-6611	6	201
1839	NORRIS	718/384-1767	12	718
1111	RHODES	201/812-1837	7	201
1352	RIVERS	718/383-3345	12	718
1332	STEPHENSON	203/675-1497	9	203
1443	WELLS	203/781-5546	11	203

ID Statement with VAR

The *ID and VAR statements* can be used together.

```
proc print data=cert.reps;
   id idnum lastname;
   var idnum sex jobcode salary;
run;
```

IDnum	LastName	IDnum	Sex	JobCode	Salary
1269	CASTON	1269	M	NA1	41690.00
1935	FERNANDEZ	1935		NA2	51081.00
1417	NEWKIRK	1417	,	NA2	52270.00
1839	NORRIS	1839	F	NA1	43433.00

Columns in both statements will be repeated.

WHERE Statement (Review)

The *WHERE statement* selects observations that meet the criteria specified in the WHERE expression.

```
proc print data=cert.reps;
   var LastName FirstName City Salary;
   where city contains 'OR';
run;
```

WHERE *WHERE-expression*;

Viewing the Log

Only 6 of the 8 observations from **cert.reps** were selected by the WHERE statement.

```
25  proc print data=cert.reps;
26    var LastName FirstName City Salary;
27    where city contains 'OR';
28  run;

NOTE: There were 6 observations read from the data set CERT.REPS.
      WHERE city contains 'OR';
```

Viewing the Output

PROC PRINT Output

Obs	LastName	FirstName	City	Salary
1	CASTON	FRANKLIN	STAMFORD	41690.00
2	FERNANDEZ	KATRINA	BRIDGEPORT	51081.00
4	NORRIS	DIANE	NEW YORK	43433.00
6	RIVERS	SIMON	NEW YORK	5379.80
7	STEPHENSON	ADAM	BRIDGEPORT	42178.00
8	WELLS	AGNES	STAMFORD	422.74

original observation numbers

Controlling Which Observations Are Read

- By default, SAS processes every observation in a SAS data set, from the first observation to the last.
- The FIRSTOBS= and OBS= data set options can be used to control which observations are processed.
- The FIRSTOBS= and OBS= options are used with input data sets on DATA and PROC steps.
- You cannot use either option with output data sets.

These temporarily override system option values FIRSTOBS=1 and OBS=max

The OBS= Data Set Option

The OBS= data set option specifies an ending point for processing an input data set.

General form of OBS= data set option:

SAS-data-set(OBS=*n*)

This option specifies the number of the last observation to process, not how many observations should be processed.

Using the OBS= Data Set Option

This OBS= data set option causes the DATA step to stop processing after observation 100.

```
data australia;
   set orion.employee_addresses (obs=100);
   if Country='AU' then output;
run;
```

Partial SAS Log

```
NOTE: There were 100 observations read from the data set
      ORION.EMPLOYEE_ADDRESSES.
NOTE: The data set WORK.AUSTRALIA has 24 observations and
      9 variables.
```

The FIRSTOBS= Data Set Option

The FIRSTOBS= data set option specifies a starting point for processing an input data set. This option specifies the number of the first observation to process.

General form of the FIRSTOBS= data set option:

SAS-data-set (FIRSTOBS=*n*)

FIRSTOBS= and OBS= are often used together to define a range of observations to be processed.

Using OBS= and FIRSTOBS= Data Set Options

The FIRSTOBS= and OBS= data set options cause the SET statement below to read 51 observations from **orion.employee_addresses**. Processing begins with observation 50 and ends after observation 100.

```
data australia;
   set orion.employee_addresses
       (firstobs=50 obs=100);
   if Country='AU' then output;
run;
```

Check the SAS Log

Partial SAS Log

```
640  data australia;
641      set orion.employee_addresses(firstobs=50 obs=100);
642      if Country='AU' then output;
643  run;

NOTE: There were 51 observations read from the data set
      ORION.EMPLOYEE_ADDRESSES.
NOTE: The data set WORK.AUSTRALIA has 13 observations and
      9 variables.
```

Using OBS= and FIRSTOBS= in a PROC Step

The FIRSTOBS= and OBS= data set options can also be used in SAS procedures. The PROC PRINT step below begins processing at observation 10 and ends after observation 15.

```
proc print data=cert.heart (firstobs=10 obs=15);
run;
```

Check the Output

Partial SAS Log

```
29    proc print data=cert.heart (firstobs=10 obs=15);
30    run;
NOTE: There were 6 observations read from the data set
      CERT.HEART.
```

PROC PRINT Output

Obs	Patient	Sex	Survive	Shock	Arterial	Heart	Cardiac	Urinary
10	509	2	SURV	OTHER	79	84	256	90
11	742	1	DIED	HYPOVOL	100	54	135	0
12	609	2	DIED	NONSHOCK	93	101	260	90
13	318	2	DIED	OTHER	72	81	410	405
14	412	1	SURV	BACTER	61	87	296	44
15	601	1	DIED	BACTER	84	101	260	377

original observation numbers

Adding a WHERE Statement

When the FIRSTOBS= or OBS= option and the WHERE statement are used together, the following occurs:

- the subsetting WHERE is applied first
- the FIRSTOBS= and OBS= options are applied to the resulting observations.

The following step includes a WHERE statement and an OBS= option.

```
proc print data=sashelp.cars
           (obs=10);
   where origin='Asia';
   var Make Model MSRP Origin;
run;
```

Check the Output

Partial SAS Log

```
31    proc print data=sashelp.cars(obs=10);
32       where origin='Asia';
32       var Make Model MSRP Origin;
34    run;

NOTE: There were 10 observations read from the data set SASHELP.CARS.
      WHERE origin='Asia';
```

The WHERE statement is applied first, and then 10 observations are processed.

PROC PRINT Output

Obs	Make	Model	MSRP	Origin
1	Acura	MDX	$36,945	Asia
2	Acura	RSX Type S 2dr	$23,820	Asia
3	Acura	TSX 4dr	$26,990	Asia
4	Acura	TL 4dr	$33,195	Asia
5	Acura	3.5 RL 4dr	$43,755	Asia
6	Acura	3.5 RL w/Navigation 4dr	$46,100	Asia
7	Acura	NSX coupe 2dr manual S	$89,765	Asia
150	Honda	Civic Hybrid 4dr manual (gas/electric)	$20,140	Asia
151	Honda	Insight 2dr (gas/electric)	$19,110	Asia
152	Honda	Pilot LX	$27,560	Asia

The subsetting variable does not need to be included in the report.

SUM Statement

The *SUM statement* calculates and displays report totals for the requested ***numeric*** variables.

```
proc print data=cert.insure;
   var Name Policy;
   where pctinsured=50;
   sum BalanceDue Total;
run;
```

SUM *variable(s)*;

PROC PRINT Output

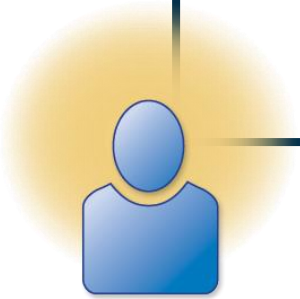
Obs	Name	Policy	BalanceDue	Total
4	Johnson, R	39022	61.04	122.07
5	LaMance, K	63265	43.68	87.35
10	Eberhardt, S	81589	173.17	346.33
14	Quigley, M	97048	99.01	198.01
15	Cameron, L	42351	111.41	222.82
			488.31	976.58

Producing Basic Reports

Sorting and Grouping Data

Business Scenario

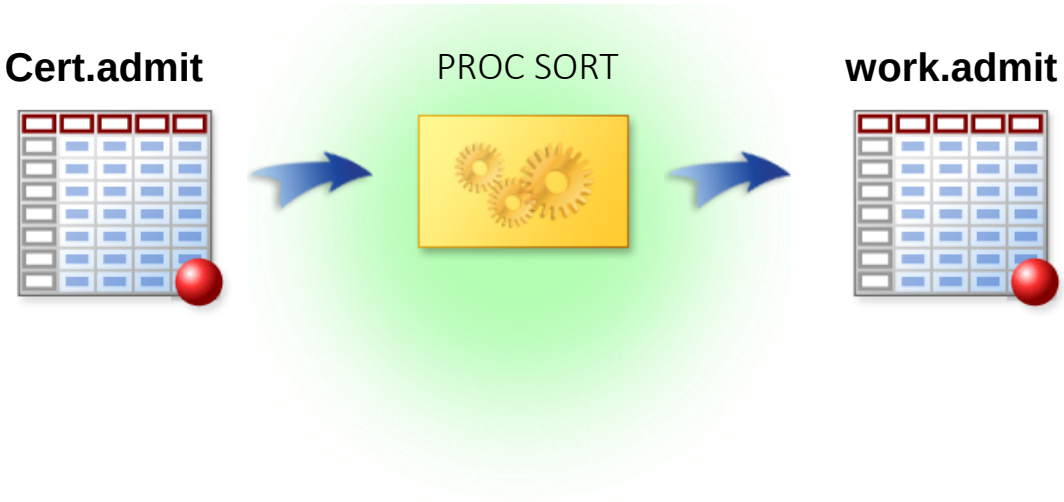
Display observations from **cert.admit** in ascending order by the variable **actlevel**.



actlevel	age	height	weight
xxxx	99	99	999
xxxx	99	99	999
xxxx	99	99	999

Creating a Sorted Report

Step 1	Use the SORT procedure to create a new data set, **work.admit**. Order the observations by the value of **actlevel**.



Creating a Sorted Report

Step 2	Use the PRINT procedure to display the sorted data set, **work.admit**.

Step 1: SORT Procedure

The *SORT procedure* rearranges the observations in the input data set based on the values of the variable or variables listed in the BY statement.

```
proc sort data=cert.admit
                 out=work.admit;
   by actlevel;
run;
```

PROC SORT DATA=*input-SAS-data-set*
<OUT=*output-SAS-data-set*>;
BY <DESCENDING> *variables*;
RUN;

The BY statement in a PROC SORT step specifies the sort variables, and if you indicate it, the sort order.

SORT Procedure

The SORT procedure

- replaces the original data set or creates a new one
- can sort on multiple variables
- sorts in ascending (default) or descending order
- does not generate printed output.

The input data set is overwritten unless the OUT= option is used to specify an output data set.

Viewing the Log

The SORT procedure does not produce a report. Check the log for errors or warnings.

Partial SAS Log

```
34   proc sort data=cert.admit
35             out=work.admit;
36      by actlevel;
37   run;

NOTE: There were 21 observations read from the data set
CERT.ADMIT.
NOTE: The data set WORK.ADMIT has 21 observations and 9
variables.
```

Step 2: Viewing the Output

```
proc print data=work.admit (firstobs=7 obs=15);
   var actlevel age height weight;
run;
```

PROC PRINT Output

Obs	ActLevel	Age	Height	Weight
7	HIGH	41	67	141
8	LOW	31	61	123
9	LOW	51	71	158
10	LOW	34	73	154
11	LOW	49	64	172
12	LOW	28	62	118
13	LOW	60	71	191
14	LOW	22	63	139
15	MOD	43	63	137

Short Answer Poll

Which step sorts the observations in a SAS data set and overwrites the same data set?

a.
```
proc sort data=work.EmpsAU
          out=work.sorted;
   by First;
run;
```

b.
```
proc sort data=orion.EmpsAU
          out=EmpsAU;
   by First;
run;
```

c.
```
proc sort data=work.EmpsAU;
   by First;
run;
```

Short Answer Poll – Correct Answer

Which step sorts the observations in a SAS data set and overwrites the same data set?

a.
```
proc sort data=work.EmpsAU
          out=work.sorted;
   by First;
run;
```

b.
```
proc sort data=orion.EmpsAU
          out=EmpsAU;
   by First;
run;
```

c.
```
proc sort data=work.EmpsAU;
   by First;
run;
```

Business Scenario

Produce a report that lists therapy patients grouped by **ActLevel**, in descending **Age** order within ActLevel.

ActLevel=HIGH

Obs	ActLevel	Age	Height	Weight
1	HIGH	44	66	140
2	HIGH	41	67	141
3	HIGH	40	69	163
4	HIGH	34	66	152
5	HIGH	29	76	193
6	HIGH	27	72	168
7	HIGH	25	75	188

ActLevel=LOW

Obs	ActLevel	Age	Height	Weight
8	LOW	60	71	191
9	LOW	51	71	158
10	LOW	49	64	172

Creating a Grouped Report

Step 1	Use the SORT procedure to group data in a data set. This scenario requires two variables to be sorted: ■ **ActLevel** ■ descending **Age** within **ActLevel**
Step 2	Use a BY statement in PROC PRINT to display the sorted observations grouped by **ActLevel**.

Step 1: Sort the Data

Sort the data set to group the observations.

```
proc sort data=cert.admit
                out=work.admit;
   by ActLevel descending Age;
run;
```

BY <DESCENDING> *variables*;

Specifying Sort Order

The *DESCENDING option* reverses the sort order for the variable that immediately follows it. The observations are sorted from the largest value to the smallest value.

Examples:

```
by descending Last
First;
```

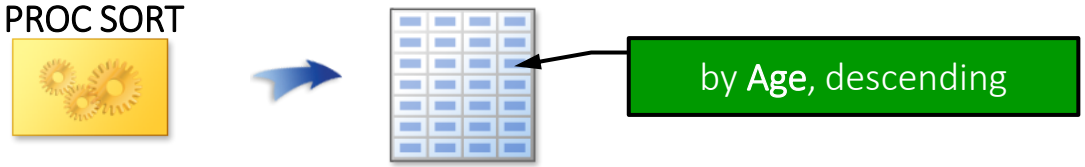
```
by Last descending
First;
```

```
by descending Last descending First;
```

Specifying Multiple BY Variables

- PROC SORT first arranges the data set by the values of the first BY variable.

- PROC SORT then arranges any observations that have the same value as the first BY variable by the values of the second BY variable.



- This sorting continues for every specified BY variable.

Step 2: Specify Report Groupings

The BY statement in a PROC PRINT step specifies the variable or variables to use to form *BY groups*.

```
proc print data=work.admit (obs=10);
         var actlevel age height weight;
         by actlevel;
run;
```

BY <DESCENDING> *variables*;

- The variables in the BY statement are called *BY variables*.
- The observations in the data set ***must*** be in order according to the order of the BY variable (or variables).

Viewing the Output

PROC PRINT Output

ActLevel=HIGH

Obs	ActLevel	Age	Height	Weight
1	HIGH	44	66	140
2	HIGH	41	67	141
3	HIGH	40	69	163
4	HIGH	34	66	152
5	HIGH	29	76	193
6	HIGH	27	72	168
7	HIGH	25	75	188

ActLevel=LOW

Obs	ActLevel	Age	Height	Weight
8	LOW	60	71	191
9	LOW	51	71	158
10	LOW	49	64	172

Short Answer Poll

Why does this program fail?

```
proc sort data=orion.sales
          out=work.sorted;
     by Country Gender;
run;

proc print data=work.sorted;
     by Gender;
run;
```

Short Answer Poll – Correct Answer

Why does this program fail?

The input data set was not sorted by Gender.

```
188  proc sort data=orion.sales
189            out=work.sorted;
190      by Country Gender;
191  run;

NOTE: There were 165 observations read from the data set ORION.SALES.
NOTE: The data set WORK.SORTED has 165 observations and 9 variables.

192
193  proc print data=work.sorted;
194      by Gender;
195  run;

ERROR: Data set WORK.SORTED is not sorted in ascending sequence. The current
       BY group has Gender = M and the next BY group has Gender = F.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 64 observations read from the data set WORK.SORTED.
```

Business Scenario

Modify the previous report to display selected variables, the Fee subtotal for each ActLevel, and the Fee grand total.

```
-------------------- ActLevel=High -------------------

Age         Height             Weight        Fee

XXXX           XXXXXXX            X            99999
XXXX           XXXXXXX            X            99999
--------                                    ---------
ActLevel                                       999999

-------------------- ActLevel=Low --------------------

Age         Height             Weight        Fee

XXXXXXX        XXXXXXX            X             99999
XXXXXXX        XXXXXXX            X             99999
--------                                    ---------
ActLevel                                       999999
                                            =========
                                              9999999
```

subtotals

grand total

Generating Subtotals

Use a BY statement and a SUM statement in a PROC PRINT step.

```
proc sort data=cert.admit
              out=work.admit;
   by actlevel descending age;
run;

proc print data=work.admit noobs;
         var actlevel age height weight;
         by actlevel;
         sum Fee;
run;
```

Viewing the Output

ActLevel=HIGH

ActLevel	Age	Height	Weight	Fee
HIGH	44	66	140	149.75
HIGH	41	67	141	149.75
HIGH	40	69	163	124.80
HIGH	34	66	152	124.80
HIGH	29	76	193	124.80
HIGH	27	72	168	85.20
HIGH	25	75	188	85.20
ActLevel				844.30

subtotal for HIGH

ActLevel=LOW

ActLevel	Age	Height	Weight	Fee
LOW	60	71	191	149.75
LOW	51	71	158	124.80
LOW	49	64	172	124.80
LOW	34	73	154	124.80
LOW	31	61	123	149.75
LOW	28	62	118	85.20
LOW	22	63	139	85.20
ActLevel				844.30

subtotal for LOW

ActLevel=MOD

ActLevel	Age	Height	Weight	Fee
MOD	54	71	183	149.75
MOD	47	72	173	124.80
MOD	43	63	137	149.75
MOD	43	65	123	124.80
MOD	35	70	173	149.75
MOD	32	67	151	149.75
MOD	30	69	147	149.75
ActLevel				998.35
				2686.95

grand total

Generating Subtotals

Add a PAGEBY statement to print each BY group on a separate page.

```
proc sort data=cert.admit
              out=work.admit;
   by actlevel descending age;
run;

proc print data=work.admit noobs;
         var actlevel age height weight;
         by actlevel;
         sum Fee;
         pageby actlevel;
run;
```

Using PAGEBY

This demonstration illustrates the use of PAGEBY to print BY groups on separate pages.

Setup for the Poll

Modify the previous report to display only employees earning less than 25,500. Which WHERE statement (or statements) results in the most efficient processing?

```
proc sort data=orion.sales
                out=work.sales;
   /* where Salary<25500; */
   by Country descending Salary;
run;
proc print data=work.sales noobs;
   by Country;
   sum Salary;
   /*     where Salary<25500; */
   var First_Name Last_Name Gender Salary;
run;
```

Multiple Choice Poll

Which WHERE statement (or statements) results in the most efficient processing?

a. The WHERE statement in the PROC SORT step.

b. The WHERE statement in the PROC PRINT step.

c. Both WHERE statements are needed.

d. The WHERE statements are equally efficient.

4.07 Multiple Choice Poll – Correct Answer

Which WHERE statement (or statements) results in the most efficient processing?

a. The WHERE statement in the PROC SORT step.

b. The WHERE statement in the PROC PRINT step.

c. Both WHERE statements are needed.

d. The WHERE statements are equally efficient.

Subsetting in PROC SORT is more efficient. It selects and sorts only the required observations.

Be sure to use the OUT= option when you subset in PROC SORT or you will overwrite your original data set with the subset.

Business Scenario

Enhance the payroll report by adding titles, footnotes, and descriptive column headings.

Obs	Employee_ID	Last_Name	Salary
1	9999	xxxxxxxxxx	99999
2	9999	xxxxxxxxxx	99999
3	9999	xxxxxxxxxx	99999

Orion Star Sales Staff
Salary Report

Obs	Employee ID	Last Name	Annual Salary
1	9999	xxxxxxxxxx	99999
2	9999	xxxxxxxxxx	99999
3	9999	xxxxxxxxxx	99999

Confidential

LABEL Statement on a DATA Step

The LABEL statement assigns permanent descriptive labels to variables.

Sales Title

Date Hired

work.subset1

First_Name	Last_Name	Salary	Job_Title	Hire_Date	Bonus

LABEL Statement

The LABEL statement assigns descriptive labels to variables.

- A label can be up to 256 characters and include any characters, including blanks.
- Labels are used automatically by most procedures.
- The PRINT procedure uses labels only when the LABEL or SPLIT= option is specified.

Defining Permanent Labels

Use a LABEL statement in a DATA step to permanently assign labels to variables. The labels are stored in the descriptor portion of the data set.

```
data work.subset1;
   set orion.sales;
   where Country='AU' and
         Job_Title contains 'Rep';
   Bonus=Salary*.10;
   label Job_Title='Sales Title'
             Hire_Date='Date Hired';
   drop Employee_ID Gender Country
        Birth_Date;
run;
```

LABEL *variable='label '*
<variable='label '...>;

Viewing the Output

```
proc contents data=work.subset1;
run;
```

Partial PROC CONTENTS Output

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
6	Bonus	Num	8	
1	First_Name	Char	12	
5	Hire_Date	Num	8	Date Hired
4	Job_Title	Char	25	Sales Title
2	Last_Name	Char	18	
3	Salary	Num	8	

Viewing the Output: Displaying Labels

To use labels in the PRINT procedure, use the LABEL option in the PROC PRINT statement.

```
proc print data=work.subset1 label;
run;
```

Partial PROC PRINT Output

Obs	First_ Name	Last_Name	Salary	Sales Title	Date Hired	Bonus
1	Irenie	Elvish	26600	Sales Rep. II	6575	2660.0
2	Christina	Ngan	27475	Sales Rep. II	8217	2747.5
3	Kimiko	Hotstone	26190	Sales Rep. I	10866	2619.0
4	Lucian	Daymond	26480	Sales Rep. I	8460	2648.0
5	Fong	Hofmeister	32040	Sales Rep. IV	8460	3204.0

LABEL Statement and PROC Print Option

Use a LABEL statement and the LABEL option to display temporary descriptive column headings instead of variable names.

```
title1 'Orion Star Sales Staff';
title2 'Salary Report';
footnote1 'Confidential';

proc print data=orion.sales label;
   var Employee_ID Last_Name Salary;
   label Employee_ID='Sales ID'
             Last_Name='Last Name'
             Salary='Annual Salary';
run;

title;
footnote;
```

LABEL *variable-1*='*label*'

...

variable-n='*label*';

Viewing the Output

Orion Star Sales Staff
Salary Report

Obs	Sales ID	Last Name	Annual Salary
1	120102	Zhou	108255
2	120103	Dawes	87975
3	120121	Elvish	26600
...			
164	121144	Capachietti	83505
165	121145	Lansberry	84260

Confidential

SPLIT= Option

The SPLIT= option in PROC PRINT specifies a split character to control line breaks in column headings.

```
proc print data=orion.sales split='*';
   var Employee_ID Last_Name Salary;
   label Employee_ID='Sales ID'
         Last_Name='Last*Name'
         Salary='Annual*Salary';
run;
```

SPLIT='*split-character*'

The SPLIT= option can be used instead of the LABEL option in a PROC PRINT step.

Viewing the Output

Partial PROC PRINT Output

```
                Orion Star Sales Staff
                     Salary Report

                                Last              Annual
Obs          Sales ID           Name              Salary

  1            120102           Zhou              108255
  2            120103           Dawes              87975
  3            120121           Elvish             26600
...
164            121144           Capachietti        83505
165            121145           Lansberry          84260

                     Confidential
```

Enhancing Variables with Labels and Formats

- A label changes the way a column name appears when displayed or printed.
- A format changes the way a value appears when displayed or printed.
- Neither change the actual underlying data.

Creating Custom Formats

Creating and Using Custom Formats – Chapter 12

Formatting Data Values

Name	Gender	Age	Height	Weight	Birthdate
Alfred	M	14	69	112.5	16370
Alice	F	13	56.5	84	16756
Barbara	F	13	65.3	98	16451
Carol	F	14	62.8	102.5	16256

Name	Gender	Age	Height	Weight	Birthdate
Alfred	M	14	69	113	26OCT2004
Alice	F	13	57	84	16NOV2005
Barbara	F	13	65	98	15JAN2005
Carol	F	14	63	103	04JUL2004

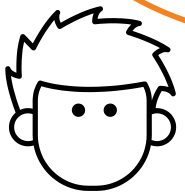
```
proc print data=pg2.class_birthdate noobs;
    format Height Weight 3.0 Birthdate date9.;
run;
```

Formatting Data Values

Name	Gender	Height
James	M	57.3
Jane	F	59.8
John	M	59
Louise	F	56.3
Robert	M	64.8

Name	Gender	Height
James	Male	Below Average
Jane	Female	Average
John	Male	Average
Louise	Female	Below Average
Robert	Male	Above Average

SAS doesn't always have a predefined format that meets your needs.



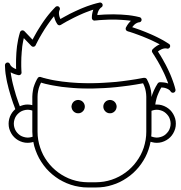
```
format Gender ? Height ?;
```

FORMAT Procedure

```
PROC FORMAT;
      VALUE format-name  value-or-range-1 = 'formatted-value'
                         value-or-range-2 = 'formatted-value'
                         . . . ;
RUN;
```

- The name can be up to 32 characters in length.
- Character formats must begin with a $ followed by a letter or underscore.
- Numeric formats must begin with a letter or underscore.
- The name cannot end in a number or match an existing SAS format.

You can use the FORMAT procedure to create your own format.



FORMAT Procedure

```
PROC FORMAT;
        VALUE format-name  value-or-range-1 = 'formatted-value'
                           value-or-range-2 = 'formatted-value'
                           . . . ;
RUN;
```

individual value or range of values that you want to format

format that you want to apply to the individual value or range of values

Creating and Using Custom Formats

create format

```
proc format;
    value $genfmt 'F'='Female'
                  'M'='Male';
run;
```

no period in format name

apply format

```
proc print data=pg2.class_birthdate;
    format Gender $genfmt.;
run;
```

period in format name

Creating and Using Custom Formats

The demonstration illustrates using the FORMAT procedure to create custom numeric and character formats based on single values and a range of values.

Defining a Continuous Range

Put < before the ending value in a range to exclude the value.

```
value hrange 50-<58 = 'Below Average'
             58-60  = 'Average'
             60<-70 = 'Above Average';
```

Put < after the starting value in a range to exclude the value.

Using Keywords

lowest possible value

```
value hrange low-<58   = 'Below Average'
              58-60    = 'Average'
             60<-high  = 'Above Average';
```

highest possible value

```
value $genfmt 'F' = 'Female'
              'M' = 'Male'
            other = 'Miscoded';
```

all values that do not match any other value