COMPBL (string)

— Returns a character value with all multiple blanks in the string converted to single blanks.

COMPRESS (string <, characters>)

— Returns a character value with specified characters removed from the string.

STRIP (string)

↳ Returns a character value with leading & trailing blanks removed from the string.

Extracting words from a String

City = scan (Location, 1);

The new column City is assigned the same length as Location.

The SCAN function treats the following characters as delimiters:

blank

! .

$ /

% ;

& <

(^

) |

*

+

,

-

Searching for Characters in Strings

The FIND function returns a number that represents the first character position where substring is found in string

FIND(string, substring, <'modifiers'>)

I = case insensitive search

T = trim leading & trailing blanks from string & substring

There are many functions do identify the location of related characters

LENGTH(string)

ANYDIGIT(string)

ANYALPHA(string)

ANYPUNCT(string)

LENGTHC

LENGTHN

ANYLOWER

ANYUPPER

ANYSPACE

Replacing Character Strings

TRANWRD (source , target , replacement)

character column

TRANSLATE

TRANSTRN

Building Character Strings

you can use concatenation functions to combine text strings or numbers into a single string.

CAT (S1 ... , Sn)

CATS (S1 ... , Sn)

CATX (' delimiter ' , S1 ... S2)

Use the END= option in the SET statement do create a temporary variable in the PDV named LastRow. Add an IF-THEN statement do write the value of MaxLength do the log if the value of LastRow is 1.

```
data parklookup;

set pg2.np_unstructured_code
    end = LastRow;

...

if LastRow = 1 then putlog
    MaxLength= ;

run;
```

Using Special Functions to Convert Column Type

↳ What happens when you try to perform an action on a column that isn't the proper type?

The informat specifies how the character value looks so that it can be converted to a numeric value.

INPUT(source, informat)

↳ Converts character values to numeric values using a specified informat.

An informat is used to indicate how the character string should be read.

The Date column has been defined as character.

Dates are best stored as numeric values that reference the number of days since January 1, 1960.

```
Date2 = input(Date, date9.);
```

The multipurpose ANYDTDTE informat can read dates written in many ways.

ANYDTDTEw.

If a date is ambiguous, such as 6/1/2018 which can be interpreted as either June 1 or January 6.

SAS uses the DATESTYLE = system option to determine the sequence.

The default value for the DATESTYLE = option is determined by the value of the LOCALE = system option.

For example, if LOCALE = is English, then the DATESTYLE = order is MDY, so 6/1 would be read as June 1.

Be careful not to specify a decimal value with the informat unless you want to insert a new decimal point.

Converting the Type of an Existing Column

The SET statement reads Volume into the PDV as a character column.

PDV

Date	...	Volume
$9		$12

When the SET statement reads the pg2.stocks data, the PDV is established & Volume is a character column.

Converting the Type of an Existing Column.

```
set pg2.stock2(rename=(Volume=charVolume));
```

Rname the input column that you want to change.

Volume will be read into the PDV as charVolume.

```
Volume=input(charVolume,comma12.);
```

Use the INPUT function on the renamed column to create a column with the original name.

This adds Volumn do the PDV as a numeric column.

```
Day=put(Date,downame3.);
```

rame

format

The format tells SAS how do display the new character value.

Formats for converting Numeric to Character:

DATE9.

DOWNAME3.

YEAR4.

COMMA10.2

DOLLAR11.2

6.2

The format specifies how the numeric value should look as a character value.

ANYDTDTEw.

* SAS automatically tries to convert character values to numeric values using the w. informat.

* SAS automatically tries to convert numeric values to character values using the BEST12. format.

* If SAS automatically converts the data, a note is displayed in the SAS log.

* If you explicitly tell SAS to convert the data with a function, a note is not displayed in the SAS log.

* Some functions such as the CAT functions automatically convert data from numeric to character & also remove leading blanks on the converted data. No note is displayed in the SAS log.

→

```
drop ____ Precip;
if Precip ne 'T' then
        PrecipNum = input(Precip, 6.);
else PrecipNum = 0;
TotalPrecip + PrecipNum;
```

The CAT functions automatically convert numeric values to character values & remove leading blanks in the converted value. SAS doesn't write a note to the log when values are converted with the CAT functions.

When SAS automatically converts a numeric value to a character value, the BEST12. format is used & the resulting character value is right-aligned. The numeric value of 30320 becomes the character value of seven leading spaces followed by 30320.

```
ZipCodeLast2 = substr(put(ZipCode, z5.), 4, 2);
```

```
proc print data=pg2.np_lodging
            (obs=10);
  where CL2010 >0;
run;

data stays;
set pg2.np_lodging;
Stay1 = largest(1, of CL:);
StayAvg = round(mean(of CL:));
if StayAvg > 0;
keep Park Stay:;
format Stay: comma11.;
run;
```

```
data rainsummary;
set pg2.np_hourlyrain;
by Month;
if first.Month=1 then
        MonthlyRainTotal=0;
MonthlyRainTotal + Rain;
if last.Month=1;
Date= datepart(DateTime);
MonthEnd = intnx('month',
                 Date,
                 0,
                 'end');
```

```
proc sort
data=pg2.np_weather
    (keep= Name Code Date Snow)
out = winter2015_2016;
where date between
        '01Oct15'd and
        '01Jun16'd and
        Snow>0;
by Code Date;
run;
```

A function returns a value that might be used in an assignment statement or expression

```
Mean2 = mean(Quiz1 - Quiz5)
```

Without the OF keyword, Mean2 is the mean of the difference between Quiz1 & Quiz5

```
data rainsummary;
set pg2.np_hourlyrain;
by Month;
if first.Month=1 then
        MonthlyRainTotal=0;
MonthlyRainTotal + Rain;
if last.Month=1;
Date= datepart(DateTime);
MonthEnd = intnx('month',
        Date,
        0,
        'end');
```

```
proc sort
data=pg2.np_weather
    (keep=Name Code Date Snow)
out = winter2015_2016;
where date between
        '01Oct15'd and
        '01Jun16'd and
        Snow>0;
by Code Date;
run;
```

A function returns a value that might be used in an assignment statement or expression

```
Mean2 = mean(Quiz1 — Quiz5)
```

Without the OF keyword, Mean2 is the mean of the difference between Quiz1 & Quiz5

The values appear the same, but the function changes the stored values, whereas the format affects only the displayed values.

10July2018 05Sep2018

Months2Pay = intck('month', ServiceDate, PayDate);

Two end-of-month boundaries were crossed at the end of July & August.

One month boundary was crossed at August 10.

The next boundary will not occur until September 10.

The STRIP function removes leading & trailing blanks.

CAT function → trailing blanks are included in the concatenated string.

CATS function → no trailing blanks are removed in the concatenated string.

Volume cannot be in the PDV as both numeric & character.

The character attribute of Volume from the SET statement is controlling the column type.

→

```
set pg2.stocks2(rename =
(Volume = CharVolume Date = CharDate));
```

L4 : ★ Creating Custom Formats

★ Creating Custom Formats from Tables.

format Weight 3.0

middle of 3 & no decimal places

MONYY7.

COMMA12.

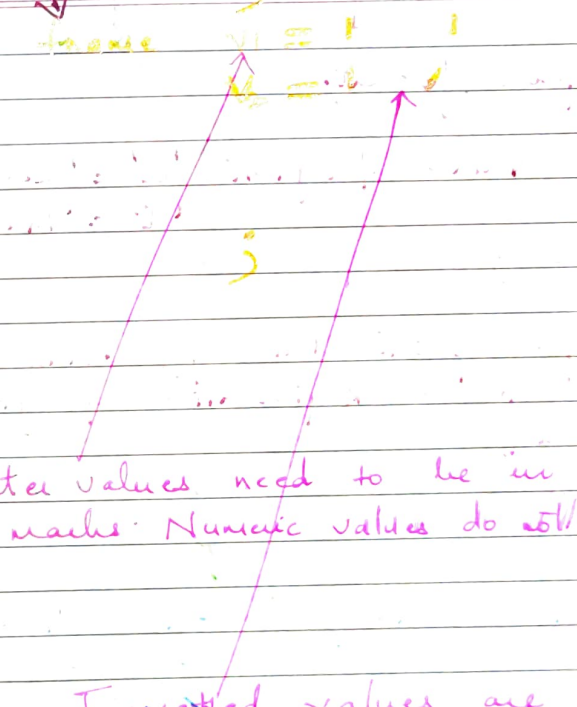
DOLLAR8.2

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

PROC FORMAT;

VALUE

$ => for character else numeric formats



Character values need to be in quotation marks. Numeric values do not

Formatted values are enclosed in quotation marks

* You can define more than 1 custom format at a time by using multiple VALUE statements in PROC FORMAT

Creating & Using Custom Formats

all values that doesn't match any other value.

```
proc format;
  value $regfmt 'C' = 'Complete'
                'I' = 'Incomplete'
                other = 'Miscoded';
run;

proc print data=pg2.class_birthdate;
  format Registration $regfmt.;
run;
```

lowest possible value

Defining a Continuous Range

Put < before the existing value in a range to exclude the value

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

highest possible value

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

The LOW keyword includes

missing values for character variables & doesn't include missing values for numeric variables.

```
proc format;
  value stdate
    low - '31DEC1999'd = '1999 & before'
    '01JAN2000'd - '31DEC2009'd
        = '2000 to 2009'
    '01JAN2010'd - high
        = '2010 and later'
    . = 'Not Supplied';

proc freq data=pg2.storm_summary;
  tables Basin * StartDate;
  format StartDate stdate.
         Basin $region.;
run;
```

Creating Custom Formats From Tables.

Rules for the Input Table

FmtName	Start	Label

Name of the custom format

Value to format

Label that you want to apply to value

The input table must have at least these three columns.

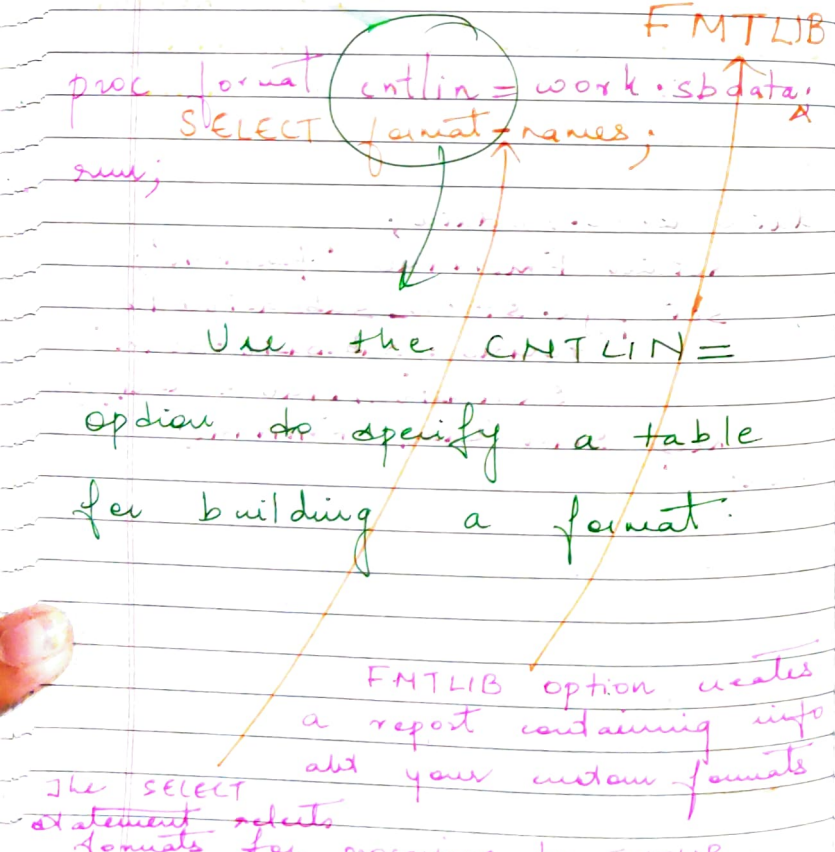
To build the table to generate the format, the RETAIN statement creates the FmtName column & retains the value $'sbfmt' for each row.

```
data work.sbdata;
    retain FmtName '$sbfmt';
    set pg2.storm_subbasincodes(
        rename=(Sub_Basin=Start
                SubBasin_Name=Label));
    Keep Start Label FmtName;
run;
```

It is very simple to read a table to create a custom format when the table has the correct layout.

Simply use the CNTLIN= option in the PROC FORMAT statement to name the input table.

CNTLIN = Option



Location of Custom Formats

By default, custom formats are stored in the temporary work library in a catalog named formats.

Format $STTYPE has been output
Format $STTYPE is already on the library WORK.FORMATS

Creating Permanent Custom Formats

LIB=

Use the LIBRARY= option to save formats in a permanent location.

PG2.MYFMTS

library → catalog name

If you specify only the library, the default catalog formats is used.

Searching for Custom Formats

options fmtsearch=(pg2.myfmts sashelp);

Work. formats → Library. formats → pg2. myfmts → sashelp. formats

proc format library=pg2.myfmt;

options fmtsearch=(pg2.formats);

Updating a Custom Format by Using the CNTLOUT = Option

1. Add a FORMAT statement in the DATA step to format the values of the four numeric columns.
2. Add a FORMAT statement in the PROC MEANS steps to format the values of Date.

...

Formats that you use in the DATA step are permanent attributes that are stored in the descriptor portion of the table.

Formats that you use in a PROC step are temporary attributes

The length of the format matches the length of the longest label.

Combining Tables

Concatenating Tables with Matching Columns

```
DATA O;
    SET I1 I2...;
RUN
```

using variables of input table

Use the SET statement when you want to combine tables with similar data in one output table.

Concatenating Tables with Matching Columns

rows from second table added after rows from the first table.

```
data storm_complete;
    set pg2.storm_summary
        pg2.storm_2017 (rename=
            (Year = Season) drop=Location);
...
```

Setting Column Attributes

```
data class_current;
    set sashelp.class
        pg2.class_new2 (rename=
            (Student = Name));
run;
```

* If columns with the same name in multiple tables have a different type (character or numeric), the DATA step fails.

PDY

Name	...other columns...
$8	

Column lengths are determined by the first table listed in the SET statement.

Merging Columns

LENGTH statement comes before the SET statement so that it will establish the attributes of Name in the PDV

```
data class_current;
   length Name $ 9;
   set sashelp.class;
   pg2.class_new (rename = (student = Name));
run;
```

PDV

Name	... other columns ...
$ 9	

Use a LENGTH statement before the SET statement do establish an appropriate length.

Merging Tables / (or joining tables)

PROC SQL → method for joining tables.

Typical the ... step preceded by ... PROC SQL;

```
DATA O;
   MERGE I1 I2 ...;
   BY BY-column(s);
RUN;
```

The input tables must be sorted by the column (or columns) listed in the BY statement

one-to-one matching

one-to-many

List any number of input tables with one or more common columns

List the common column or columns

The DATA step includes the MERGE statement, listing the sorted tables.

The BY statement identifies Name as the common column in the two tables that will be used to determine matching rows.

Merging Tables: Execution

Rows are read sequentially from both tables. When the BY values match, they are both read into the PDV.

One-to-Many Merge

The BY values match, and both rows are read into the PDV.

The RUN statement triggers an implicit output & implicit return to the top of the DATA step.

Because all columns in the PDV are read via the MERGE statement all values are automatically retained.

The BY values do not match, but one value matches the PDV. That row is read into the PDV and overwrites previous values.

When SAS recognizes that it has completed reading all values for a BY group, the PDV is reinitialized & all columns are set to missing.

The PDV is reset to missing values when a new BY group begins.

If data needs to be sorted prior to merge:

```
PROC SORT DATA=[illegible] OUT=outputT.T;
    BY [illegible];
RUN;
```

Identifying Matching & Nonmatching Rows

Merging Tables with Nonmatching Rows

The new table includes matches & nonmatches.

The BY values do not match. SAS reinitializes the PDV & reads the next row in sequence.

SAS then compares BY values in the two tables & finds that they doesn't match, so it reads the row from the table with the BY value that comes first in sorted sequence.

Sequential matching continues until all rows are read from each i/p table.

Merging Tables with Nonmatching Rows

The IN= data set option can be used to identify matching & nonmatching rows.

```
MERGE I1 (IN=variable)
      I2 (IN=variable) ...;
```

The IN= data set option creates temporary variables in the PDV that you can use to flag matching or nonmatching rows.

- The IN= variables are 0 if the BY value is not in the corresponding input table & 1 if the BY value is in the corresponding input table.

```
merge pg2.class_update(in=inUpdate)
      pg2.class_teachers(in=inTeachers);
```

Values are assigned in the PDV during execution but not written do the output table.

Columns in Both Tables

Use the RENAME=
data set option do give each column a unique name.

```
merge pg2.weather_sanfran2016
      (rename=(AvgTemp=AvgTemp2016))
      pg2.weather_sanfran2017
      (rename=(AvgTemp=AvgTemp2017));
by month;
```

The subsetting IF statement must be used because values for the IN= variables are assigned during execution

Merging Tables without a Common Column.

PROC SQL can join multiple tables without a common column in one query

PROC SQL can't write output do multiple tables in a single query

PROC SQL
→ doesn't require sorted data

```
do Year = 1 to 3;

do Year = 1 to 3 by 1;
```

Executing an iterative DO Loop. Year is incremented by 1 at the bottom of the DO loop.

An OUTPUT statement between the DO & END statements writes a row for each iteration of the DO loop.

Since there is no explicit OUTPUT statement, so implicit output at the end of the DATA step is active.

An iterative DO loop works great when you know exactly how many iterations do perform.

always executes once.

```
DO UNTIL (expression);
   ... repetitive code...
END;
```

checks the condition at the bottom of the loop.

checks the condition at the top of the loop

```
DO WHILE (expression);
   ... repetitive code...
END;
```

executes only if condition is true.

Combining Iterative & Conditional DO Loops.

The DO loop stops executing when the stop value is exceeded or the condition is met, whichever is first.

Iterative & Conditional DO Loop.

```
do Month=1 to 12 until/while
        (Savings > 5000);
   :
end;
```

Restructuring Tables

> Restructuring Data with the DATA Step.

Creating a Narrow Table with the DATA Step.

Restructuring Data with the TRANSPOSE PROCEDURE.

PROC TRANSPOSE can restructure data with simple statements