临床研究SAS高级编程

—— SAS Functions

Contents

Character

Contents

Introduction to Working with Character Variables (1)

Objectives

- In this section, you will learn how to do the following:
  - identify character variables
  - set the length of character variables
  - align character values within character variables
  - handle missing values of character variables
  - work with character variables, character constants, and character expressions in SAS program statements
  - instruct SAS to read fields that contain numbers as character variables in order to save space

Introduction to Working with Character Variables (2)

- A character variable is a variable whose value contains letters, numbers, and special characters, and whose length can be from 1 to 32,767 characters long. Character variables can be used in declarative statements, comparison statements, or assignment statements where they can be manipulated to create new character variables.

Input SAS Data Set for Examples

- Example:

```
Data departures;
    Input Country $1-9 CitisIntour 11-12 USGate $ 14-26
          ArrivalDepartureGates $28-48;
    Datalines;
    Japan         5     San Francisco        Tokyo,Osaka
    Italy         8     New York             Rome, Naples
    Australia    12     Honolulu         Sydney, Brisbane;

Proc print data=departures;
Title 'Data Set DEPARTURES';
Run;
```

In this dataset, which variable must be stored as character variables?

Output:

```
                         Data Set AIR.DEPARTURES                          1

                          Cities
    Obs     Country       InTour    USGate           ArrivalDepartureGates

     1      Japan            5      San Francisco    Tokyo, Osaka
     2      Italy            8      New York         Rome, Naples
     3      Australia       12      Honolulu         Sydney, Brisbane
```

Identifying Character Variables and Expressing Character Values (1)

- How to create a character variable:
  - Define it in an INPUT statement
    - Example: input Country $1-9;
  - Create a character variable and assign a value to it in an assignment statement. Simply enclose the value in quotation marks:
    - Example: Schedule='3-4 tours per person';
  - Either single quotation marks (apostrophes) or double quotation marks are acceptable. If the value itself contains a single quote, then surround the value with double quotation marks , as in
    - Example: Remarks ="See last year's schedule";
    - *Note:* Matching quotation marks properly is important. Missing or extraneous quotation marks cause SAS to misread both the erroneous statement and the statements following it.

Identifying Character Variables and Expressing Character Values (2)

Character variable property:

- When specifying a character value in an expression, you must also enclose the value in quotation marks.
  - Example: if USGate = ’San Francisco’ then Airport = ’SFO’;
- In character values, SAS distinguishes uppercase letters from lowercase letters.
  - Example: if USGate = ’Honolulu’ then Airport = ’HNL’;

    is not same with the following statement

    if USGate = ’HONOLULU’ then Airport = ’HNL’;

Identifying Character Variables and Expressing Character Values (3)

- The NOOBS option in the PROC PRINT statement suppresses the display of observation numbers in the output.
  - Example:

```
Data departures;
    Input Country $1-9 CitiesIntour 11-12 USGate $ 14-26
             ArrivalDepartureGates $28-48;
    Datalines;
    Japan       5   San Francisco          Tokyo,Osaka
     Italy      8   New York              Rome, Naples
Australia      12   Honolulu          Sydney, Brisbane;

Proc print data=departures noobs;
Title 'Data Set DEPARTURES';
Run;
```

Identifying Character Variables and Expressing Character Values (4)

Output

```
                    Tours By City of Departure                          1

Country          Schedule                Remarks            USGate        Airport

Japan     3-4 tours per season See last year's schedule San Francisco    SFO
Italy     3-4 tours per season See last year's schedule New York
Australia 3-4 tours per season See last year's schedule Honolulu         HNL
```

Setting the Length of Character Variables (1)

- How SAS assigns lengths to character variables
  - If SAS cannot determine a length for a character variable: 8 bytes
  - The first value for a SAS character variable determines the variable’s length.
  - LENGTH statement: before any other reference to the variable in the DATA step
    - A later use of the LENGTH statement will not change its size.
- Reducing the length of character data with the LENGTH statement

Setting the Length of Character Variables (2)

- Example:

```
data aircode;
    set mylib.departures;
    if USGate = 'San Francisco' then Airport = 'SFO';
    else if USGate = 'Honolulu' then Airport = 'HNL';
    else if USGate = 'New York' then Airport = 'JFK or LGA';
run;
proc print data=aircode;
     var Country USGate Airport;
     title 'Country by US Point of Departure';
run;
```

SAS listing output:

```
                Country by US Point of Departure                       1

           Obs    Country      USGate           Airport

            1     Japan        San Francisco     SFO
            2     Italy        New York          JFK
            3     Australia    Honolulu          HNL
```

Setting the Length of Character Variables (3)

Only the characters JFK appear in the observation for New York. SAS first encounters Airport in the statement that assigns the value SFO. Therefore, SAS creates Airport with a length of three bytes and uses only the first three characters in the New York observation.

To allow space to write JFK or LGA, use a LENGTH statement as the first reference to Airport. The LENGTH statement is a *declarative statement* and has the form

LENGTH *variable-list* $ *number-of-bytes*;

Handling Missing Values (1)

- Reading missing values:
  - Example:

```
data missingval;
     length Country $ 10 TourGuide $ 10;
     input Country TourGuide;
     datalines;
Japan     Yamada
Italy     Militello
Australia Edney
Venezuela .
Brazil    Cardoso
;
proc print data=missingval;
     title 'Missing Values for Character List Input Data';
run;
```

Handling Missing Values (2)

SAS Listing output:

```
                Missing Values for Character List Data                    1

                   Obs    Country      TourGuide

                    1     Japan        Yamada
                    2     Italy        Militello
                    3     Australia    Edney
                    4     Venezuela
                    5     Brazil       Cardoso
```

SAS recognized the period as a missing value in the fourth data line; therefore, it recorded a missing value for the character variable TourGuide in the resulting data set.

Handling Missing Values (3)

- Checking for missing character values
  - When you want to check for missing character values, compare the character variable to a blank surrounded by quotation marks:
    - Example: if USGate = ' ' then GateInformation = 'Missing';
- Setting a character variable value to missing
  - You can assign missing character values in assignment statements by setting the character variable to a blank surrounded by quotation marks.
    - For example, the following statement sets the day of departure based on the number of days in the tour. If the number of cities in the tour is a week or less, then the day of departure is a Sunday. Otherwise, the day of departure is not known and is set to a missing value.

Handling Missing Values (4)

- Example:

```
data departuredays;
      set mylib.departures;
      length DayOfDeparture $ 8;
      if CitiesInTour <=7 then DayOfDeparture = 'Sunday';
      else DayOfDeparture = ' ';
run;
proc print data=departuredays;
      var Country CitiesInTour DayOfDeparture;
      title 'Departure Day is Sunday or Missing';
run;
```

Handling Missing Values (5)

SAS lisiting output:

```
           Departure Day is Sunday or Missing                    1

                                 Cities      DayOf
        Obs     Country          InTour     Departure

         1      Japan               5        Sunday
         2      Italy               8
         3      Australia          12
         4      Venezuela           4        Sunday
```

Creating New Character Values (1)

- Creating new character values
  - The SCAN function returns a character string when it is given the source string, the position of the desired character string, and a character delimiter:

    SCAN (*source*,*n*<,*list-of-delimiters*>)

  - The LEFT function produces a value that has all leading blanks in the *source* moved to the right side of the value; therefore, the result is left aligned. The source can be any kind of character expression, including a character variable, a character constant enclosed in quotation marks, or another character function.

    LEFT (*source*)

Creating New Character Values (2)

- Example: `ArrivalGate = scan(ArrivalDepartureGates,1,',');`

- Example:

```
DepartureGate =  left(scan(ArrivalDepartureGates,2,','));
```

output:

Arrival and Departure Gates 1

Obs	Country	ArrivalDepartureGates	ArrivalGate	Departure Gate
1	Japan	Tokyo, Osaka	Tokyo	Osaka
2	Italy	Rome, Naples	Rome	Naples
3	Australia	Sydney, Brisbane	Sydney	Brisbane
4	Venezuela	Caracas, Maracaibo	Caracas	Maracaibo
5	Brazil	Rio de Janeiro, Belem	Rio de Janeiro	Belem

Creating New Character Values (3)

- Saving storage space when using the SCAN function
  - The SCAN function causes SAS to assign a length of 200 bytes to the target variable in an assignment statement. Most of the other character functions cause the target to have the same length as the original value.

Creating New Character Values (4)

Example:

```
data gatelength;
      length ArrivalGate $ 14 DepartureGate $ 9;
      set mylib.departures;
      ArrivalGate = scan(ArrivalDepartureGate,1,',');
      DepartureGate = left(scan(ArrivalDepartureGate,2,','));
run;
```

Note:

- In the data set GATELENGTH, the variable ArrivalGate has a length of 200 because the SCAN function creates it. The variable DepartureGate also has a length of 200 because the argument of the LEFT function contains the SCAN function.
- Setting the lengths of ArrivalGate and DepartureGate to the needed values rather than to the default length saves a lot of storage space. Because SAS sets the length of a character variable the first time SAS encounters it, the LENGTH statement must appear before the assignment statements that create values for the variables.

Combining Character Values: Using Concatenation (1)

Understanding concatenation of variable values

- Concatenation combines character values by placing them one after the other and assigning them to a variable. The length of the new variable is the sum of the lengths of the pieces or number of characters that is specified in a LENGTH statement for the new variable.

Combining Character Values: Using Concatenation (2)

- Example: AllGates = USGate || ArrivalDepartureGates;

```
                         All Tour Gates                                  1

        Obs     Country       USGate           ArrivalDepartureGates

         1      Japan         San Francisco    Tokyo, Osaka
         2      Italy         New York         Rome, Naples
         3      Australia     Honolulu         Sydney, Brisbane
         4      Venezuela     Miami            Caracas, Maracaibo
         5      Brazil                         Rio de Janeiro, Belem

        Obs                   AllGates

         1      San FranciscoTokyo, Osaka
         2      New York      Rome, Naples
         3      Honolulu   ❶  Sydney, Brisbane
         4      Miami         Caracas, Maracaibo
         5      ❷              Rio de Janeiro, Belem
```

❶ the middle of AllGates contain blanks?

❷ the beginning of AllGates in the Brazil observation contain blanks?

Combining Character Values: Using Concatenation (3)

Performing a simple concatenation

- The following statement combines all the cities named as gateways into a single variable named AllGates:
  - AllGates = USGate || ArrivalDepartureGates;
- SAS attaches the beginning of each value of ArrivalDepartureGates to the end of each value of USGate and assigns the results to AllGates. The following DATA step includes this statement:

Combining Character Values: Using Concatenation (4)

- Example:

```
/* first try */
options pagesize=60 linesize=80 pageno=1 nodate;
data all;
     set mylib.departures;
     AllGates = USGate || ArrivalDepartureGates;
run;
proc print data=all;
     var Country USGate ArrivalDepartureGates AllGates;
     title 'All Tour Gates';
run;
```

Combining Character Values: Using Concatenation (5)

SAS output:

```
                         All Tour Gates                                 1

         Obs     Country       USGate           ArrivalDepartureGates

          1      Japan         San Francisco    Tokyo, Osaka
          2      Italy         New York         Rome, Naples
          3      Australia     Honolulu         Sydney, Brisbane
          4      Venezuela     Miami            Caracas, Maracaibo
          5      Brazil                         Rio de Janeiro, Belem

         Obs                   AllGates

          1      San FranciscoTokyo, Osaka
          2      New York      Rome, Naples
          3      Honolulu  ❶    Sydney, Brisbane
          4      Miami         Caracas, Maracaibo
          5      ❷               Rio de Janeiro, Belem
```

❶ the middle of AllGates contain blanks?

❷ the beginning of AllGates in the Brazil observation contain blanks?

Combining Character Values: Using Concatenation (6)

Removing interior blanks

- The TRIM function produces a value without the trailing blanks in the source.
  - Example: `AllGate2 = trim(USGate) || ArrivalDepartureGates;`

The following program adds this statement to the DATA step:

```
/* removing interior blanks */
options pagesize=60 linesize=80 pageno=1 nodate;
data all2;
   set mylib.departures;
   AllGate2 = trim(USGate) || ArrivalDepartureGates;
run;
proc print data=all2;
   var Country USGate ArrivalDepartureGates AllGate2;
   title 'All Tour Gates';
run;
```

Combining Character Values: Using Concatenation (7)

SAS listing output

```
                          All Tour Gates                                   1

Obs   Country     USGate          ArrivalDepartureGates          AllGate2

 1    Japan       San Francisco   Tokyo, Osaka          San FranciscoTokyo, Osaka
 2    Italy       New York        Rome, Naples          New YorkRome, Naples
 3    Australia   Honolulu        Sydney, Brisbane      HonoluluSydney, Brisbane
 4    Venezuela   Miami           Caracas, Maracaibo    MiamiCaracas, Maracaibo
 5    Brazil                      Rio de Janeiro, Belem  Rio de Janeiro, Belem
                                                        ❶
```

Notice at ❶ that the AllGate2 value for Brazil has a blank space before Rio de Janeiro, Belem. When the TRIM function encounters a missing value in the argument, one blank space is returned. In this observation, USGate has a missing value;

Combining Character Values: Using Concatenation (8)

Adding additional characters

- Data set ALL2 shows that removing the trailing blanks from USGate causes all the values of ArrivalDepartureGates to appear immediately after the corresponding values of USGate. To make the result easier to read, you can concatenate a comma and blank between the trimmed value of USGate and the value of ArrivalDepartureGates. Also, to align the AllGate3 value for Brazil with all other values of AllGate3, use an IF-THEN statement to equate the value of AllGate3 with the value of ArrivalDepartureGates in that observation.
  - Example:

```
AllGate3 = trim(USGate)||', '||ArrivalDepartureGates;
if Country = 'Brazil' then AllGate3 = ArrivalDepartureGates;
```

Combining Character Values: Using Concatenation (9)

- Example

```
/* final version */
options pagesize=60 linesize=80 pageno=1 nodate;
data all3;
     set mylib.departures;
     AllGate3 = trim(USGate)||', '||ArrivalDepartureGates;
     if Country = 'Brazil' then AllGate3 = ArrivalDepartureGates;
run;
proc print data=all3;
   var Country USGate ArrivalDepartureGates AllGate3;
   title 'All Tour Gates';
run;
```

Combining Character Values: Using Concatenation (10)

Sas listing output

```
                               All Tour Gates                                   1

Obs Country    USGate        ArrivalDepartureGates          AllGate3

 1  Japan      San Francisco Tokyo, Osaka          San Francisco, Tokyo, Osaka
 2  Italy      New York      Rome, Naples          New York, Rome, Naples
 3  Australia  Honolulu      Sydney, Brisbane      Honolulu, Sydney, Brisbane
 4  Venezuela  Miami         Caracas, Maracaibo    Miami, Caracas, Maracaibo
 5  Brazil                   Rio de Janeiro, Belem Rio de Janeiro, Belem
```

Combining Character Values: Using Concatenation (11)

- Troubleshooting: When new variables appear truncated
  - When concatenating variables, you might see the apparent loss of part of a concatenated value. Earlier in this section, ArrivalDepartureGates was divided into two new variables, ArrivalGate and DepartureGate, each with a default length of 200 bytes. (Remember that when a variable is created by an expression that uses the SCAN function, the variable length is 200 bytes.) For reference, this example re-creates the DATA step:

Combining Character Values: Using Concatenation (12)

- Example:

```
options pagesize=60 linesize=80 pageno=1 nodate;
data gates;
     set mylib.departures;
     ArrivalGate = scan(ArrivalDepartureGates,1,',');
     DepartureGate = left(scan(ArrivalDepartureGates,2,','));
run;
```

Note: If the variables ArrivalGate and DepartureGate are concatenated, as they are in the next DATA step, then the length of the resulting concatenation is 402 bytes: 200 bytes for each variable and 1 byte each for the comma and the blank space. This example uses the VLENGTH function to show the length of ADGates.

Combining Character Values: Using Concatenation (13)

- Example:

```
 /* accidentally omitting the TRIM function */
options pagesize=60 linesize=80 pageno=1 nodate;
data gates2;
   set gates;
   ADGates = ArrivalGate||', '||DepartureGate;;
   ADLength = vlength(ADGates);
run;
proc print data=gates2;
   var Country ArrivalDepartureGates ADGates ADLength;
   title 'All Tour Gates';
run;
```

Combining Character Values: Using Concatenation (14)

SAS listing output:

```
          All Tour Gates         1

Obs Country      ArrivalDepartureGates

 1  Japan        Tokyo, Osaka
 2  Italy        Rome, Naples
 3  Australia    Sydney, Brisbane
 4  Venezuela    Caracas, Maracaibo
 5  Brazil       Rio de Janeiro, Belem

Obs                                     ADGates

 1  Tokyo
 2  Rome
 3  Sydney
 4  Caracas
 5  Rio de Janeiro

Obs ADLength

 1     402
 2     402
 3     402
 4     402
 5     402
```

Combining Character Values: Using Concatenation (15)

- The concatenated value from DepartureGate appears to be truncated in the output.It has been concatenated after the trailing blanks of ArrivalGate, and it does not appear because the output does not display 402 bytes.
  - The TRIM function can trim the trailing blanks from ArrivalGate, as shown in the preceding section. The significant characters from all three pieces that are assigned to ADGates can then fit in the output.
  - The length of ADGates remains 402 bytes. The LENGTH statement can assign to the variable a length that is shorter but large enough to contain the significant pieces.

Combining Character Values: Using Concatenation (16)

- The following DATA step uses the TRIM function and the LENGTH statement to remove interior blanks from the concatenation:

```
options pagesize=60 linesize=80 pageno=1 nodate;
data gates3;
     length ADGates $ 30;
     set gates;
     ADGates = trim(ArrivalGate)||', '||DepartureGate;
run;
proc print data=gates3;
   var country ArrivalDepartureGates ADGates;
   title 'All Tour Gates';
run;
```

The following output displays the results:

Combining Character Values: Using Concatenation (17)

```
                          All Tour Gates                              1

     Obs    Country      ArrivalDepartureGates    ADGates

      1     Japan        Tokyo, Osaka             Tokyo, Osaka
      2     Italy        Rome, Naples             Rome, Naples
      3     Australia    Sydney, Brisbane         Sydney, Brisbane
      4     Venezuela    Caracas, Maracaibo       Caracas, Maracaibo
      5     Brazil       Rio de Janeiro, Belem    Rio de Janeiro, Belem
```

More Property (1)

Character values: This section illustrates the flexibility that SAS provides for manipulating character values. In addition to the functions that are described in this section,

The following character functions are also frequently used:

- COMPBL
  - removes multiple blanks from a character string.
  - Example:

SAS Statements	Results
	----+----1----+-----2--
`string='Hey Diddle   Diddle'; string=compbl(string); put string;`	`Hey Diddle Diddle`
`string='125     E Main St'; length address $10; address=compbl(string); put address;`	`125 E Main`

More Property (2)

- COMPRESS
  - Removes specified character(s) from the source
    - Example 1: Compressing Blanks

SAS Statements	Results
	`----+----1`
`a='AB C D ';`	
`b=compress(a);`	
`put b;`	`ABCD`

    - Example 2: Compressing Lowercase Letters

SAS Statements	Results
	`----+----1----+----2----+----3`
`x='123-4567-8901 B 234-5678-9012 c';`	
`y=compress(x,'ABCD','l');`	`123-4567-8901 234-5678-9012`
`put y;`	

More Property (3)

- Example 3: Compressing Tab Characters

SAS Statements	Results
	----+----1

`x='1    2    3    4    5';`
`y=compress(x,,'s');`
`put y;`
	12345

- Example 4: Keeping Characters in the List

SAS Statements	Results
	----+----1

`x='Math A English B Physics A';`
`y=compress(x,'ABCD','k');`
`put y;`
	ABA

More Property (4)

- INDEX
  - Searches the source data for a pattern of characters
    - Example

SAS Statements	Results
`a='ABC.DEF (X=Y)';`	
`b='X=Y';`	
`x=index(a,b);`	
`put x;`	10

More Property (5)

- LOWCASE
  - converts all uppercase letters to lowercase letters
    - Example

SAS Statements	Results
`x='INTRODUCTION';`	
`y=lowcase(x);`	
`put y;`	`introduction`

More Property (6)

- RIGHT
  - Right aligns a character expression
    - Example

SAS Statements	Results
	`----+----1----+`
`a='Due Date  ';`	
`b=right(a);`	
`put a $10.;`	`Due Date`
`put b $10.;`	`  Due Date`

More Property (7)

- SUBSTR
  - extracts a group of characters
    - Example

SAS Statements	Results
	`----+----1----+----2`
`date='06MAY98';`	
`month=substr(date,3,3);`	
`year=substr(date,6,2);`	
`put @1 month @5 year;`	`MAY 98`

More Property (8)

- TRANSLATE
  - Replaces specific characters in a character expression
    - Example

SAS Statements	Results
`x=translate('XYZW','AB','VW');` `put x;`	XYZB

More Property (9)

- UPCASE
  - returns the source data in uppercase.
    - Example

SAS Statements	Results
`name=upcase('John B. Smith');`	
`put name;`	`JOHN B. SMITH`

Numeric

Contents

- Introduction to working with numeric variables
- Input SAS data set for examples
- Calculating with numeric variables
- Comparing numeric variables
- Storing numeric variables efficiently
- More property

Introduction to Working with Numeric Variables

Objectives

- In this section, you will learn the following:
  - how to perform arithmetic calculations in SAS using arithmetic operators and the SAS functions ROUND and SUM
  - how to compare numeric variables using logical operators
  - how to store numeric variables efficiently when disk space is limited

Input SAS Data Set for Examples

- A numeric variable is a variable whose values are numbers.

```
data populartours;
   input country $1-11 Nights Aircost LandCost vender $;
   datalines;
   Japan        8   982    1020      Express
   Greece      12    .      748      Express
   Italy        8   852     598      Express
run;
Proc print data= populartours;
   Title 'Data Set populartours';
Run;
```

A numeric variable is a variable whose values are numbers.

In populartours,the variables Nights, AirCost, and LandCost Contain numbers and are stored as numeric variables.

Data Set populartours

Data Set MYLIB.POPULARTOURS 1

Obs	Country	Nights	Air Cost	Land Cost	Vendor
1	Japan	8	982	1020	Express
2	Greece	12	.	748	Express
3	New Zealand	16	1368	1539	Southsea

Calculating with Numeric Variables (1)

Using arithmetic operators in assignment statements

➤ One way to perform calculations on numeric variables is to write an assignment statement using arithmetic operators. Arithmetic operators indicate addition, subtraction, multiplication, division, and exponentiation (raising to a power).

Operators in Arithmetic Expressions

Operation	Symbol	Example
addition	+	x = y + z;
subtraction	–	x = y - z;
multiplication	*	x = y * z
division	/	x = y / z
exponentiation	**	x = y ** z

Calculating with Numeric Variables (2)

- Example:

```
data newtour;
       set populartours;
       TotalCost=AirCost+LandCost;
       PeakAir=(AirCost*1.10)+8;
       NightCost=LandCost/Nights;
run;
proc print data=newtour;
      var Country Nights AirCost LandCost TotalCost PeakAir  NightCost;
      title 'Costs of Tours';
run;
```

SAS Listing Output:

Costs for Tours 1

Obs	Country	Nights	Air Cost	Land Cost	Total Cost	Peak Air	Night Cost
1	Japan	8	982	1020	2002	1088.2	127.500
2	Greece	12	.	748	.	.	62.333
3	New Zealand	16	1368	1539	2907	1512.8	96.188

Calculating with Numeric Variables (3)

Understanding numeric expressions and assignment statements

- Numeric expression in SAS share some features with mathematical expressions:
  - When an expression contains more than one operator, the operations have the same order of precedence as in a mathematical expression: exponentiation is done first, then multiplication and division, and finally addition and subtraction.
  - When operators of equal precedence appear, the operations are performed from left to right (except exponentiation, which is performed right to left).
  - Parentheses are used to group parts of an expression; as in mathematical expressions, operations in parentheses are performed first.

- *Note:* The equal sign in an assignment statement does not perform the same function as the equal sign in a mathematical equation. The sequence *variable*= in an assignment statement defines the statement, and the variable must appear on the left side of the equal sign. You cannot switch the positions of the result variable and the expression as you can in a mathematical equation.

Calculating with Numeric Variables (4)

- Understanding how SAS handles missing values
  - Why SAS Assigns Missing Values

What if an observation lacks a value for a particular numeric variable? For example, in the data set MYLIB.POPULARTOURS, as shown in Output 7.2, the observation for Greece has no value for the variable AirCost. To maintain the rectangular structure of a SAS data set, SAS assigns a missing value to the variable in that observation. A missing value indicates that no information is present for the variable in that observation.

Calculating with Numeric Variables (5)

- Rules for missing values
  - The following rules describe missing values in several situations:
    - In data lines, a missing numeric value is represented by a period, for example,
      - **Greece 8 12 . 748 Express**
      - By default, SAS interprets a single period in a numeric field as a missing value. (If the INPUT statement reads the value from particular columns, as in column input, a field that contains only blanks also produces a missing value.)
    - In an expression, a missing numeric value is represented by a period, for example,
      - if AirCost= . then Status = 'Need air cost';
    - In a comparison and in sorting, a missing numeric value is a lower value than any other numeric value.
    - In procedure output, SAS by default represents a missing numeric value with a period.
    - Some procedures eliminate missing values from their analyses; others do not.
    - Documentation for individual procedures describes how each procedure handles missing values.

Calculating with Numeric Variables (6)

Propagating missing values

- When you use a missing value in an arithmetic expression, SAS sets the result of the expression to missing. If you use that result in another expression, the next result is also missing. In SAS, this method of treating missing values is called *propagation of missing values*. For example, Output 7.2 shows that in the data set NEWTOUR, the values for TOTALCOST and PEAKAIR are also missing in the observation for Greece.

Note: SAS enables you to distinguish between various kinds of numeric missing values. See “Missing Values” section of *SAS Language Reference: Concepts*. The SAS language contains 27 special missing values based on the letters A–Z and the underscore (_).

Calculating Numbers Using SAS Functions (1)

- Rounding values
  - In the example data that lists costs of the different tours (Output 7.1), some of the tours have odd prices: $748 instead of $750, $1299 instead of $1300, and so on. Rounded numbers, created by rounding the tour prices to the nearest $10, would be easier to work with. Programming a rounding calculation with only the arithmetic operators is a lengthy process. However, SAS contains around 280 built-in numeric expressions called *functions*. You can use them in expressions just as you do the arithmetic operators. For example, the following assignment statement rounds the value of AirCost to the nearest $50:
    - RoundAir = round(AirCost,50);
  - The following statement calculates the total cost of each tour, rounded to the nearest $100:
    - TotalCostR = round(AirCost + LandCost,100);

Calculating Numbers Using SAS Functions (2)

Calculating a cost when there are missing values

- As another example, the travel agent can calculate a total cost for the tours based on all nonmissing costs. Therefore, when the airfare is missing (as it is for Greece) the total cost represents the land cost, not a missing value. (Of course, you must decide whether skipping missing values in a particular calculation is a good idea.) The SUM function calculates the sum of its arguments, ignoring missing values.

  - Example: `SumCost = sum(AirCost,LandCost);`

Calculating Numbers Using SAS Functions (3)

Combining functions

- It is possible for you to combine functions. The ROUND function rounds the quantity given in the first argument to the nearest unit given in the the second argument. The SUM function adds any number of arguments, ignoring missing values. The calculation in the following assignment statement rounds the sum of all nonmissing airfares and land costs to the nearest $100 and assigns the value to RoundSum:
  - Example: `RoundSum=round(sum(AirCost,LandCost),100);`

Calculating Numbers Using SAS Functions (4)

Using the ROUND and SUM functions in the following DATA step creates the data set MORETOUR:

```
data moretour;
     set mylib.populartours;
     RoundAir = round(AirCost,50);
     TotalCostR = round(AirCost + LandCost,100);
     CostSum = sum(AirCost,LandCost);
     RoundSum = round(sum(AirCost,LandCost),100);
run;

proc print data=moretour;
     var Country AirCost LandCost RoundAir TotalCostR CostSum RoundSum;
     title 'Rounding and Summing Values';
run;
```

Calculating Numbers Using SAS Functions (5)

SAS listing output:

Rounding and Summing Values 1

Obs	Country	Air Cost	Land Cost	Round Air	Total CostR	Cost Sum	Round Sum
1	Japan	982	1020	1000	2000	2002	2000
2	Greece	.	748	.	.	748	700
3	New Zealand	1368	1539	1350	2900	2907	2900

Comparing Numeric Variables (1)

To compare two numeric variables, you can write an IF-THEN/ELSE statement using logical operators. The following table lists some of the logical operators you can use for variable comparisons.

Logical Operators

Symbol	Mnemonic Equivalent	Logical Operation
=	eq	equal
¬=, ^=, ~=	ne	not equal to (the ¬=, ^=, or ~= symbol, depending on your keyboard)
>	gt	greater than
>=	ge	greater than or equal to
<	lt	less than
<=	le	less than or equal to

Comparing Numeric Variables (2)

```
data toursunder2K;
     set populartours;
     TotalCost = AirCost + LandCost;
     if TotalCost gt 2000 then delete;
run;
proc print data=toursunder2K;
     var Country Nights AirCost Landcost TotalCost Vendor;
     title 'Tours $2000 or Less';
run;
```

```
                         Tours $2000 or Less                          1

                                          Air     Land    Total
     Obs    Country         Nights       Cost     Cost     Cost     Vendor

      1     Greece            12           .       748       .      Express
      2     Ireland            7         787       628     1415     Express
      3     Venezuela          9         426       505      931     Mundial
```

Comparing Numeric Variables (3)

The TotalCost value for Greece is a missing value because any calculation that includes a missing value results in a missing value. In a comparison, missing numeric values are lower than any other numeric value.

If you need to compare a variable to more than one value, you can include multiple comparisons in a *condition*. To eliminate tours with missing values, a second comparison is added:

Comparing Numeric Variables (4)

```
data toursunder2K2;
     set mylib.populartours;
     TotalCost = AirCost + LandCost;
      if TotalCost gt 2000 or Totalcost = . then delete;
run;
proc print data=toursunder2K2;
     var Country Nights TotalCost Vendor;
     title 'Tours $2000 or Less';
run;
```

SAS listing output:

```
                    Tours $2000 or Less                        1

                                                Total
       Obs    Country           Nights          Cost      Vendor

        1     Ireland              7            1415     Express
        2     Venezuela            9             931     Mundial
        3     Italy                8            1450     Express
```

Storing Numeric Variables Efficiently (1)

The data sets shown in this section are very small, but data sets are often very large.If you have a large data set, you may need to think about the storage space that yourdata set occupies. There are ways to save space when you store numeric variables in SAS data sets.

By default, SAS uses 8 bytes of storage in a data set for each numeric variable. Therefore, storing the variables for each observation in the earlier data set MORETOUR requires 75 bytes:

56 bytes for numeric variables
(8 bytes per variable * 7 numeric variables)
11 bytes for Country
8 bytes for Vendor

---

75 bytes for all variables

Storing Numeric Variables Efficiently (2)

When numeric variables contain only integers (whole numbers), you can often shorten them in the data set being created. For example, a length of 4 bytes accurately stores all integers up to at least 2,000,000.

A LENGTH statement contains the names of the variables followed by the number of bytes to be used for their storage. For numeric variables, the LENGTH statement affects only the data set being created; it does not affect the program data vector. The
following program changes the storage space for all numeric variables that are in the data set SHORTER:

Storing Numeric Variables Efficiently (3)

- Example:

```
data shorter;
    set mylib.populartours;
    length Nights AirCost LandCost RoundAir TotalCostR Costsum RoundSum 4;
    RoundAir = round(AirCost,50);
    TotalCostR = round(AirCost + LandCost,100);
    CostSum = sum(AirCost,LandCost);
    RoundSum = round(sum(AirCost,LandCost),100);
run;
```

By calculating the storage space that is needed for the variables in each observation of SHORTER, you can see how the LENGTH statement changes the amount of storage space used:

28 bytes for numeric variables
(4 bytes per variable in the LENGTH statement X 7 numeric variables)
11 bytes for Country
8 bytes for Vendor

---

47 bytes for all variables

Storing Numeric Variables Efficiently (4)

Because of the 7 variables in SHORTER are shortened by the LENGTH statement, the storage space for the variables in each observation is reduced by almost half.

CAUTION:

Be careful in shortening the length of numeric variables if your variable values are not integers. Fractional numbers lose precision permanently if they are truncated. In general, use the LENGTH statement to truncate values only when disk space is limited. Use the default length of 8 bytes to store variables containing fractions.

More properties (1)

Numeric:

- **Abs（x）** Returns the absolute value
  - Example: `x=abs(-3) Result=3;`

- **Mod（x1,x2）** Returns the remainder from the division of the first argument by the second argument, fuzzed to avoid most unexpected floating-point results
  - Example: `x1=mod(10,3); put x1 9.4; Result=1;`

- **Ceil:** Returns the smallest integer that is greater than or equal to the argument, fuzzed to avoid unexpected floating-point results
  - Example: `var1=2.1; a=ceil(var1); put a; Result=3;`

More properties (2)

- Floor: Returns the largest integer that is less than or equal to the argument, fuzzed to avoid unexpected floating-point results
  - Example: `var1=2.1; a=floor(var1); put a; Result=2;`

- Int: Returns the integer value, fuzzed to avoid unexpected floating-point results
  - Example: `var1=2.1; x=int(var1); put x; Result=2;`

Date and Time

Contents

Introduction to Working with Dates

Objective

- SAS stores dates as single, unique numbers so that they can be used in programs like any other numeric variable. In this section you will learn how to do the following:
  - make SAS read dates in raw data files and store them as SAS date values indicate which calendar form SAS should use to display SAS date values
  - calculate with dates, that is, determine the number of days between dates, find the day of the week on which a date falls, and use today's date in calculations

Understanding How SAS Handles Dates (1)

How SAS stores date values

- Dates are written in many different ways. Some dates contain only numbers, while others contain various combinations of numbers, letters, and characters. For example, all the following forms represent the date July 26, 2000:

072600	26JUL00	002607
7/26/00	26JUL2000	July 26, 2000

- With so many different forms of dates, there must be some common ground, a way to store dates and use them in calculations, regardless of how dates are entered or displayed.
- The common ground that SAS uses to represent dates is called a *SAS date value*. No matter which form you use to write a date, SAS can convert and store that date as the number of days between January 1, 1960, and the date that you enter.

Understanding How SAS Handles Dates (2)

In SAS, every date is a unique number on a number line. Dates before January 1,1960, are negative numbers; those after January 1, 1960, are positive. Because SAS date values are numeric variables, you can sort them easily, determine time intervals, and use dates as constants, as arguments in SAS functions, or in calculations.

Note: SAS date values are valid for dates based on the Gregorian calendar from A.D. 1582 through A.D. 19,900. Use caution when working with historical dates. Although the Gregorian calendar was used throughout most of Europe from 1582, Great Britain and the American colonies did not adopt the calendar until 1752.

Input File and SAS Data Set for Examples

Example:

①	②	③
Japan	13may2000	8
Greece	17oct99	12
New Zealand	03feb2001	16
Brazil	28feb2001	8
Venezuela	10nov00	9
Italy	25apr2001	8
USSR	03jun1997	14
Australia	24oct98	12
Ireland	27aug2000	7

① the name of country toured

② the departure date

③ the number of nights on the tour

Entering Dates (1)

Understanding informats for date values

- In order for SAS to read a value as a SAS date value, you must give it a set of directions called an informat. By default, SAS reads numeric variables with a standard numeric informat that does not include letters or special characters. When a field that contains data does not match the standard patterns, you specify the appropriate informat in the INPUT statement.
- Four commonly used informates are:
  - MMDDYY8. reads dates written as *mm/dd/yy*.
  - MMDDYY10. reads dates written as *mm/dd/yyyy*.
  - DATE7. reads dates in the form *ddMMMyy*.
  - DATE9. reads dates in the form *ddMMMyyyy*.

Entering Dates (2)

Reading a date value

- To create a SAS data set for the Tradewinds Travel data, the DATE9. informat is used in the INPUT statement to read the variable DepartureDate.
  - input Country $ 1-11 @13 DepartureDate date9. Nights;
- Using an informat in the INPUT statement is called *formatted input*. The formatted input in this example contains the following items:
  - a pointer to indicate the column in which the value begins (@13)
  - the name of the variable to be read (DepartureDate)
  - the name of the informat to use (DATE9.)

Entering Dates (3)

Example:

```
libname mylib 'permanent-data-library';
data mylib.tourdates;
     infile 'input-file';
     input Country $ 1-11 @13 DepartureDate date9. Nights;
run;

proc print data=mylib.tourdates;
     title 'Tour Departure Dates as SAS Date Values';
run;
```

Entering Dates (4)

SAS listing output

```
                    Tour Departure Dates as SAS Date Values                   1

                                          Departure
                   Obs     Country           Date        Nights

                     1     Japan            14743           8
                     2     Greece           14534          12
                     3     New Zealand      15009          16
                     4     Brazil           15034           8
                     5     Venezuela        14924           9
                     6     Italy            15090           8
                     7     Russia           13668          14
                     8     Switzerland      14989           9
                     9     Australia        14176          12
                    10     Ireland          14849           7
```

Entering Dates (5)

- Using good programming practices to read dates
  - When reading dates, it is good programming practice to always use the DATE9. or MMDDYY10. informats to be sure that the data is read correctly.

Entering Dates (6)

- Example:

```
data mylib.tourdates7;
   infile 'input-file';
   input Country $ 1-11 @13 DepartureDate date7. Nights;
run;
proc print data=mylib.tourdates7;
   title 'Tour Departure Dates Using the DATE7. Informat';
   title2 'Displayed as Two-Digit Calendar Dates';
   format DepartureDate date7.;
run;
proc print data=mylib.tourdates7;
   title 'Tour Departure Dates Using the DATE7. Informat';
   title2 'Displayed as Four-Digit Calendar Dates';
   format DepartureDate date9.;
run;
```

Entering Dates (7)

Obs	Country	Departure Date ①	Nights ②
1	Japan	13MAY20	0
2	Greece	17OCT99	12
3	New Zealand	03FEB20	1
4	Brazil	28FEB20	1
5	Venezuela	10NOV00	9
6	Italy	25APR20	1
7	USSR	14JAN20	1
8	Australia	24OCT98	12
9	Ireland	27AUG20	0

Obs	Country	Departure Date ③	Nights
1	Japan	13MAY1920	0
2	Greece	17OCT9999	12
3	New Zealand	03FEB1920	1
4	Brazil	28FEB1920	1
5	Venezuela	10NOV2000	9
6	Italy	25APR1920	1
7	USSR	14JAN1920	14
8	Australia	24OCT1998	12
9	Ireland	27AUG1920	7

① SAS stopped reading the date after seven characters; it read the first two digits, the century, and not the complete four-digit year.

② To read the data for the next variable, SAS moved the pointer one column and read the next two numeric characters (the years 00, 01, and 97) as the value for the variable Nights. The data for Nights in the input file was ignored.

③ When the dates were formatted for four-digit calendar dates, SAS used the YEARCUTOFF= 1920 system option to determine the century for the two-digit year. What was originally 1997 in observation 7 became 2019, and what was originally 2000 and 2001 in observations 1, 3, 4, 6, 8, and 10 became 1920.

Entering Dates (8)

Using dates as constants

- To write a SAS date constant, enclose a date in quotation marks in the standard SAS form *ddMMMyyyy* and immediately follow the final quotation mark with the letter D. The D suffix tells SAS to convert the calendar date to a SAS date value.

  - Example:

```
if Country = 'Switzerland' then DepartureDate = '21jan2001'd;
```

Displaying Dates (1)

Understanding how SAS displays values

- SAS displays all data values with a set of directions called a format. By default, SAS uses a standard numeric format with no commas, letters, or other special notation to display the values of numeric variables.

Displaying Dates (2)

- Formatting a date value
  - The format can be used by specifying the variable and the format name in a FORMAT statement.
  - The FORMAT contains the following items:
    - The name of the variable
    - The name of the format to be used
  - Placing a FORMAT statement in a PROC step associates the format with the variable only for step. To associate a format with a variable permanently, use the FORMAT statement in a DATA step.

  - Example (1):

```
proc print data=mylib.tourdates;
     title 'Departure Dates in Two-Digit Calendar Format';
     format DepartureDate mmddyy8.;
run;
proc print data=mylib.tourdates;
     title 'Departure Dates in Four-Digit Calendar Format';
     format DepartureDate mmddyy10.;
run;
```

Displaying Dates (3)

Obs	Country	Departure Date	Nights
1	Japan	13MAY1920	8
2	Greece	17OCT9999	12
3	New Zealand	03FEB1920	16
4	Brazil	28FEB1920	8
5	Venezuela	10NOV2000	9
6	Italy	25APR1920	8
7	USSR	14JAN1920	14
8	Australia	24OCT1998	12
9	Ireland	27AUG1920	7

Obs	Country	Departure Date	Nights
1	Japan	13MAY1920	8
2	Greece	17OCT9999	12
3	New Zealand	03FEB1920	16
4	Brazil	28FEB1920	8
5	Venezuela	10NOV2000	9
6	Italy	25APR1920	8
7	USSR	14JAN1920	14
8	Australia	24OCT1998	12
9	Ireland	27AUG1920	7

Displaying Dates (4)

Example (2)

```
data mylib.fmttourdate;
   set mylib.tourdates;
   format DepartureDate date9.;
run;
proc contents data=mylib.fmttourdate nodetails;
run;
```

Displaying Dates (5)

Output

```
                              The SAS System                                1

                         The CONTENTS Procedure

Data Set Name: MYLIB.FMTTOURDATE                      Observations:         10
Member Type:   DATA                                   Variables:            3
Engine:        V8                                     Indexes:              0
Created:       14:15 Friday, November 19, 1999        Observation Length:   32
Last Modified: 14:15 Friday, November 19, 1999        Deleted Observations: 0
Protection:                                           Compressed:           NO
Data Set Type:                                        Sorted:               NO
Label:


                -----Engine/Host Dependent Information-----

Data Set Page Size:          8192
Number of Data Set Pages:    1
First Data Page:             1
Max Obs per Page:            254
Obs in First Data Page:      10
Number of Data Set Repairs:  0
filename:                   /SAS_DATA_LIBRARY/fmttourdate.sas7bdat
Release Created:             8.0001M0
Host Created:                HP-UX
Inode Number:                1498874206
Access Permission:           rw-r--r--
Owner Name:                  user01
File Size (bytes):           16384


            -----Alphabetic List of Variables and Attributes-----

             #    Variable           Type    Len    Pos    Format
             -------------------------------------------------
             1    Country            Char     11     16
             2    DepartureDate      Num       8      0    DATE9.
             3    Nights             Num       8      8
```

Displaying Dates (6)

Changing formats temporarily

- When preparing a report that requires the date in a different format, the permanent format can be overrode by using a FORMAT statement in a proc step.
  - Example:

```
proc print data=mylib.tourdates;
     title 'Tour Departure Dates';
     format DepartureDate worddate18.;
run;
```

Note: The format DATE9. is still permanently assigned to DepartureDate. Calendar dates in the remaining examples are in the form *ddMMMyyyy* unless a FORMAT statement is included in the PROC PRINT step.

Output:

Obs	Country	Departure Date	Nights
1	Japan	MAY 13,1920	8
2	Greece	OCT 17,1999	12
3	New Zealand	FEB 3,1920	16
4	Brazil	FEB 28,1920	8
5	Venezuela	NOV 10,2000	9
6	Italy	APR 25,1920	8
7	USSR	JAN 14,1920	14
8	Australia	OCT 24,1998	12
9	Ireland	AUG 27,1920	7

Using Dates in Calculations (1)

Sorting dates

- Since SAS date values are numeric values, they can be sorted and used in calculations.
  - Example:

```
proc sort data=mylib.fmttourdate out=sortdate;
   by DepartureDate;
run;
proc print data=sortdate;
   var DepartureDate Country Nights;
   title 'Departure Dates Listed in Chronological Order';
run;
```

Using Dates in Calculations (2)

Creating new date variables

- In the previous example, the return date for each tour can be calculated for each tour, To start, create a new variable by adding the number of nights to the departure date, as follows:
  - Example:

```
data home;
   set mylib.tourdates;
   Return = DepartureDate + Nights;
   format Return date9.;
run;
proc print data=home;
   title 'Dates of Departure and Return';
run;
```

Obs	Country	Departure Date	Nights	Return
1	Japan	14743	8	21MAY2000
2	Greece	14534	12	29OCT1999
3	New Zealand	15009	16	19FEB2001
4	Brazil	15034	8	08MAR2001
5	Venezuela	14924	9	19NOV2000
6	Italy	15090	8	03MAY2001
8	Australia	14176	12	05NOV1998
9	Ireland	14849	7	03SEP2000

Using SAS Date Functions (1)

Finding the day of the week

- Constructing a data set with these statements produces a list of payment due dates. The following program includes these statements and assigns the format WEEKDATE29. to the new variable DueDate:

```
data pay;
   set mylib.tourdates;
   DueDate = DepartureDate - 30;
   if Weekday(DueDate) = 1 then
      DueDate = DueDate - 1;
   format DueDate weekdate29.;
run;
proc print data=pay;
   var Country DueDate;
   title 'Date and Day of Week Payment Is Due';
run;
```

Obs	Country	DueDate
1	Japan	Thursday, April 13,2000
2	Greece	Friday, September 17,1999
3	New Zealand	Thursday, January 4,2001
4	Brazil	Monday, January 29,2001
5	Venezuela	Wednesday, October11,2000
6	Italy	Monday, March 26, 2001
8	Australia	Thursday, September 24,1998
9	Ireland	Friday, July 28,2000

Using SAS Date Functions (2)

Calculating a date from today

- The TODAY function produces a SAS date value that corresponds to the date when the program is run. The following statements determine which tours depart at least 90 days from today's date but not more than 180 days from now:

```
data ads;
   set mylib.tourdates;
   Now = today();
   if Now + 90 <= DepartureDate <= Now + 180;
run;
proc print data=ads;
   title 'Tours Departing between 90 and 180 Days from Today';
   format DepartureDate Now date9.;
run;
Output:
```

Obs	Country	Departure Date	Nights	Now
1	Japan	13May2000	8	23NOV1999

北京生物统计与数据管理联合会 Beijing Biometric Association

Comparing Durations and SAS Date Values (1)

Example1:

```
/* Calculating a duration in days */
data ttage;
   Start = '08feb82'd;
   RightNow = today();
   Age = RightNow - Start;
   format Start RightNow date9.;
run;
proc print data=ttage;
   title 'Age of Tradewinds Travel';
run;
```

Output:

Obs	Start	RightNow	Age
1	08FEB1982	23NOV1999	6497

Note: The value of Age is 6497, a number that looks like an unformatted SAS date value. However, Age is actually the difference between February 8, 1982, and November 23,1999, and represents a duration in days, not a SAS date value. To make the value of Age more understandable, divide the number of days by 365 (more precisely, 365.25) to produce a duration in years. The following DATA step calculates the age of Tradewinds Travel in years:

Comparing Durations and SAS Date Values (2)

Example2:

```
/* Calculating a duration in years */
data ttage2;
   Start = '08feb82'd;
   RightNow = today();
   AgeInDays = RightNow - Start;
   AgeInYears = AgeInDays / 365.25;
   format AgeInYears 4.1 Start RightNow date9.;
run;
proc print data=ttage2;
   title 'Age in Years of Tradewinds Travel';
run;
```

Output:

Obs	Start	Rightnow	Age In Days	Age In Years
1	08FEB1982	23NOV1999	6497	17.8

Statistics

Contents

RANGE Function

Details

- The RANGE function returns the difference between the largest and the smallest of the nonmissing arguments.

Syntax

- **RANGE** (*argument,argument*,...)
  - Argument is numeric. At least one nonmissing argument is required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by OF

Example

SAS Statements	Results
`x0=range(.,.);`	.
`x1=range(-2,6,3);`	8
`x2=range(2,6,3,.);`	4
`x3=range(1,6,3,1);`	5
`x4=range(of x1-x3);`	4

RMS Function (1)

Details

- The root mean square is the square root of the arithmetic mean of the squares of the values. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the root mean square of the non-missing values.
- Let n be the number of arguments with non-missing values, and let be $x_1, x_2, \ldots, x_n$ the values of those arguments. The root mean square is

$$\sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

RMS Function (2)

Syntax

- **RMS** (*argument<,argument,…>*)
  - Argument is a non-negative numeric constant, variable, or expression.
  - Tip: The argument list can consist of a variable list, which is preceded by OF.

Example

SAS Statements	Results
`x1=rms(1,7);`	5
`x2=rms(.,1,5,11);`	7
`x3=rms(of x1-x2);`	6.0827625303

SKEWNESS Function

Syntax

- **SKEWNESS** (*argument,argument,argument, ...*)
  - *Argument* is numeric. At least three nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list may consist of a variable list, which is preceded by OF.

Example

SAS Statements	Results
`x1=skewness(0,1,1);`	-1.732050808
`x2=skewness(2,4,6,3,1);`	0.5901286564
`x3=skewness(2,0,0);`	1.7320508076
`x4=skewness(of x1-x3);`	-0.953097714

STD Function

Syntax

- **STD** (*argument,argument*,...)
  - *Argument* is numeric. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=std(2,6);`	2.8284271247
`x2=std(2,6,.);`	2.8284271427
`x3=std(2,4,6,3,1);`	1.9235384062
`x4=std(of x1-x3);`	0.5224377453

STDERR Function

Syntax

- **STDERR** (*argument,argument, ...*)
  - *Argument* is numeric. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by OF.

Example

SAS Statements	Results
`x1=stderr(2,6);`	2
`x2=stderr(2,6,.);`	2
`x3=stderr(2,4,6,3,1);`	0.8602325267
`x4=stderr(of x1-x3);`	0.3799224911

SUM Function (1)

Syntax

- **SUM** (*argument,argument*, …)
  - *Argument* is numeric. If all the arguments have missing values, the result is a missing value. The argument list can consist of a variable list, which is preceded by OF.

SUM Function (2)

Example

SAS Statements	Results
`x1=sum(4,9,3,8);`	24
`x2=sum(4,9,3,8,.);`	24

`x1=9;`
`x2=39;`
`x3=sum(of x1-x2);`
	48

`x1=5; x2=6; x3=4; x4=9;`
`y1=34; y2=12; y3=74; y4=39;`
`result=sum(of x1-x4, of y1-y5);`
	183

`x1=55;`
`x2=35;`
`x3=6;`
`x4=sum(of x1-x3, 5);`
	101

`x1=7;`
`x2=7;`
`x5=sum(x1-x2);`
	0

`y1=20;`
`y2=30;`
`x6=sum(of y:);`
	50

USS Function

Syntax

- **USS** (*argument-1*<,*argument-n*>)
  - *Argument* is numeric. At least one nonmissing argument is required. Otherwise, the function returns a missing value. If you have more than one argument, the argument list can consist of a variable list, which is preceded by OF.

Example

SAS Statements	Results
`x1=uss(4,2,3.5,6);`	68.25
`x2=uss(4,2,3.5,6,.);`	68.25
`x3=uss(of x1-x2);`	9316.125

VAR Function

Syntax

- **VAR** (*argument,argument, ...*)
  - *Argument* is numeric. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by OF.

Example

SAS Statements	Results
`x1=var(4,2,3.5,6);`	2.7291666667
`x2=var(4,6,.);`	2
`x3=var(of x1-x2);`	0.2658420139

Probability

Contents

POISSON Function

Details

- The POISSON function returns the probability that an observation from a Poisson distribution, with mean *m*, is less than or equal to *n*. To compute the probability that an observation is equal to a given value, *n*, compute the difference of two probabilities from the Poisson distribution for *n* and *n*− .

Syntax

- **POISSON** (*m*,*n*)
  - *M* is a numeric mean parameter. Range: $m \geq 0$
  - *N* is an integer random variable. Range: $n \geq 0$

Example

SAS Statements	Results
`x=poisson(1,2);`	0.9196986029

PROBBETA Function

Details

- The PROBBETA function returns the probability that an observation from a beta distribution, with shape parameters *a* and *b*, is less than or equal to *x*.

Syntax

- **PROBBETA** (*x*,*a*,*b*)
  - *X* is a numeric random variable. Range: $0 \le x \le 1$
  - *A* is a numeric shape parameter. Range: $a > 0$
  - *B* is a numeric shape parameter. Range: $b > 0$

Example

SAS Statements	Results
`x=probbeta(.2,3,4);`	0.09888

PROBBNML Function

Details

- The PROBBNML function returns the probability that an observation from a binomial distribution, with probability of success *p*, number of trials *n*, and number of successes *m*, is less than or equal to *m*. To compute the probability that an observation is equal to a given value *m*, compute the difference of two probabilities from the binomial distribution for *m* and *m*−1 successes.

Syntax

- **PROBBNML** (*p*,*n*,*m*)
  - *P* **is a numeric probability of success parameter.** RANGE: $0 \le p \le 1$
  - *N* **is an integer number of independent Bernoulli trials parameter.** RANGE: $n > 0$
  - *M* **is an integer number of successes random variable.** RANGE: $0 \le m \le n$

Example

SAS Statements	Results
`x=probbnml(0.5,10,4);`	`0.376953125`

PROBBNRM Function (1)

Details

➢ The PROBBNRM function returns the probability that an observation (X, Y) from a standardized bivariate normal distribution with mean 0, variance 1, and a correlation coefficient *r*, is less than or equal to (*x*, *y*). That is, it returns the probability that X≤*x* and Y≤*y*. The following equation describes the PROBBNRM function, where *u* and *v* represent the random variables *x* and *y*, respectively:

$$\mathrm{PROBBNRM}(x, y, r) = \frac{1}{2\pi\sqrt{1-\mathrm{r}^2}} \int_{-\infty}^{x} \int_{-\infty}^{y} \exp\left[-\frac{u^2 - 2ruv + v^2}{2\left(1-r^2\right)}\right] dv\ du$$

PROBBNRM Function (2)

Syntax

- **PROBBNRM** (*x*, *y*, *r*)
- *X* is a numeric variable.
- *Y* is a numeric variable.
- *R* is a numeric correlation coefficient. Range: $-1 \le r \le 1$

Example

SAS Statements	Result
`p=probbnrm(.4, -.3, .2);`	
`put p;`	0.2783183345

PROBCHI Function

Details

- The PROBCHI function returns the probability that an observation from a chi-square distribution, with degrees of freedom *df* and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts a noninteger degrees of freedom parameter *df*. If the optional parameter *nc* is not specified or has the value 0, the value returned is from the central chi-square distribution.

Syntax

- **PROBCHI** (*x*,*df*<,*nc*>)
  - *X* is a numeric random variable. Range: $x \geq 0$
  - *Df* is a numeric degrees of freedom parameter. Range: $df > 0$
  - *Nc* is an optional numeric noncentrality parameter.Range: $nc \geq 0$

Example:

SAS Statements	Results
`x=probchi(11.264,11);`	`0.5785813293`

PROBF Function (1)

Details

- The PROBF function returns the probability that an observation from an *F* distribution, with numerator degrees of freedom *ndf*, denominator degrees of freedom *ddf*, and noncentrality parameter *nc*, is less than or equal to *x*. The PROBF function accepts noninteger degrees of freedom parameters *ndf* and *ddf*. If the optional parameter *nc* is not specified or has the value 0, the value returned is from the central *F* distribution.
- The significance level for an *F* test statistic is given by

```
p=1-probf(x,ndf,ddf);
```

PROBF Function (2)

Syntax

- **PROBF** (*x,ndf,ddf*<,nc>)
  - *X* is a numeric random variable. Range: $x \geq 0$
  - *Ndf* is a numeric numerator degrees of freedom parameter. Range: $ndf > 0$
  - *Ddf* is a numeric denominator degrees of freedom parameter. Range: $ddf > 0$
  - *Nc* is an optional numeric noncentrality parameter. Range: $nc \geq 0$

Example

SAS Statements	Results
`x=probf(3.32,2,3);`	`0.8263933602`

PROBGAM Function

Details

- The PROBGAM function returns the probability that an observation from a gamma distribution, with shape parameter *a*, is less than or equal to *x*.

Syntax

- **PROBGAM** (*x*,*a*)
  - *X* is a numeric random variable. Range: $x \geq 0$
  - *A* is a numeric shape parameter. Range: $a > 0$

Example

SAS Statements	Results
`x=probgam(1,3);`	0.0803013971

PROBIT Function (1)

Details

- The PROBIT function returns the *p*th quantile from the standard normal distribution. The probability that an observation from the standard normal distribution is less than or equal to the returned quantile is *p*.

CAUTION:

- The result could be truncated to lie between -8.222 and 7.941.
- *Note:* PROBIT is the inverse of the PROBNORM function.

PROBIT Function (2)

Syntax

- **PROBIT** (*p*)
  - *P* is a numeric probability. Range: $0 < p < 1$

Example

SAS Statements	Results
`x=probit(.025);`	-1.959963985
`x=probit(1.e-7);`	-5.199337582

PROBNORM Function

Details

- The PROBNORM function returns the probability that an observation from the standard normal distribution is less than or equal to *x*.

Syntax

- **PROBNORM** (*x*)
  - *X* is a numeric random variable.

Example

SAS Statements	Results
`x=probnorm(1.96);`	0.9750021049

Others

Contents

- Function LARGEST, SMALLEST, ORDINAL
- Function LENGTH, LENGTHC, LENGTHN
- Function CAT, CATS, CATT, CATX
- Function SCAN, SCANQ
- Function INDEX, INDEXC, INDEXW
- Function FIND, FINDC
- Function COUNT, COUNTC

Function LARGEST SMALLEST ORDINAL

- Function **LARGEST, SMALLEST, ORDINAL**
  - Example

    largest_num=LARGEST (k, 456, 789, .Q, 123)

    smallest_num = SMALLEST (k, 456, 789, .Q, 123);

    ordinal_num = ORDINAL (k, 456, 789, .Q, 123);

k	LARGEST Function	SMALLEST Function	ORDINAL Function
1	789	123	Q
2	456	456	123
3	123	789	456
4	.	.	789

Function LENGTH, LENGTHC, LENGTHN

- **LENGTH:** returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string
- **LENGTHC:** returns the length of a character string, including trailing blanks.
- **LENGTHN:** returns the length of a non-blank character string, excluding trailing blanks, and returns 0 for a blank character string.

a	x=length(a)	y=lengthc(a)	z=lengthn(a)
‘Baboons Eat Bananas_’	19	20	19
‘’	1	1	0

Function CAT, CATS, CATT, CATX

- **CAT**: concatenates character strings without removing leading or trailing blanks.
- **CATS**: concatenates character strings and removes leading and trailing blanks.
- **CATT**: concatenates character strings and removes trailing blanks only.
- **CATX**: concatenates character strings, removes leading and trailing blanks, and inserts separators.

- Example
  - exlot1=CAT(bhnum1x,bhnum2x,bhnum3x);
  - exlot2=CATS(bhnum1x,bhnum2x,bhnum3x);
  - exlot3=CATT(bhnum1x,bhnum2x,bhnum3x);
  - exlot4=CATX(' ',bhnum1x,bhnum2x,bhnum3x);

bhnum1x	bhnum2x	bhnum3x
P6-00	P6-00	V2031
P6-00	P6-00	V2031

exlot1	exlot2	exlot3	exlot4
P6-00 P6-00 V2031	P6-00P6-00V2031	P6-00 P6-00 V2031	P6-00 P6-00 V2031
P6-00 P6-00 V2031	P6-00P6-00V2031	P6-00 P6-00 V2031	P6-00 P6-00 V2031

Function SCAN, SCANQ

- **SCAN**: Selects a given word from a character expression
- **SCANQ**: Returns the *n*th word from a character expression, ignoring delimiters that are enclosed in quotation marks

- Example:
  - allnames='Eleanor "Billie Holiday" Fagan';

i	1	2	3	4
scan(allnames,i," ")	Eleanor	"Billie	Holiday"	Fagan
scanq(allnames,i," ")	Eleanor	"Billie Holiday"	Fagan	

Function INDEX, INDEXC, INDEXW

- **INDEX:** Searches a character expression for a string of characters
- **INDEXC:** Searches a character expression for specific characters
- **INDEXW:** Searches a character expression for a specified string as a word

- Example: xyz="babc,abc@abc";

Syntax	statement	result	xyz
INDEX(source,excerpt)	index(x, "abc");	2	babc,abc@abc
INDEXC(source,excerpt-1<,… excerpt-n>)	indexc(x,"ac","b");	1	babc,abc@abc
INDEXW(source, excerpt<,delimiter>)	indexw(x,"abc","@");	10	babc,abc@abc
	indexw(x,"abc","@,");	6	babc,abc@abc

Function FIND, FINDC (1)

- **FIND** function
  - Searches for a specific substring of characters within a character string that you specify
  - Modifiers: i, t
- Example:
  - xyz=‘This is a thistle? Yes, this is a thistle.’;
  - a=‘this_’

b	c	statement	search for	result	xyz
		find(xyz,a)	this_	25	‘This is a thistle? Yes, this is a thistle.’
‘i’		find(xyz,a,b)	this_,This_	1	‘This is a thistle? Yes, this is a thistle.’
‘t’		find(xyz,a,b)	this	11	‘This is a thistle? Yes, this is a thistle.’
‘it’		find(xyz,a,b)	this,This	1	‘This is a thistle? Yes, this is a thistle.’
‘i’	30	find(xyz,a,b,c)	this_,This_	0	‘This is a thistle? Yes, this is a thistle.’
‘it’	30	find(xyz,a,b,c)	this,This	35	‘This is a thistle? Yes, this is a thistle.’

Function FIND, FINDC (2)

FINDC function

- Searches for specific characters that either appear or do not appear within a character string that you specify
- Modifiers: i, t, v

Example:

- xyz=‘Baboons Eat Bananas ’;
- a=‘ab’

b	c	statement	search for	result	xyz
		findc(xyz,a)	a,b	2	‘Baboons_Eat_Bananas_’
‘v’		findc(xyz,a,b)	^(a,b)	1	‘Baboons_Eat_Bananas_’
‘ivt’		findc(xyz,a,b)	^(a,b,A,B)	4	‘Baboons_Eat_Bananas_’
‘v’	13	findc(xyz,a,b,c)	^(a,b)	13	‘Baboons_Eat_Bananas_’
‘ivt’	13	findc(xyz,a,b,c)	^(a,b,A,B)	15	‘Baboons_Eat_Bananas_’

Function COUNT, COUNTC (1)

- **COUNT** function
  - Counts the number of times that a specific substring of characters appears within a character string that you specify
  - Modifiers: i, t

- Example:
  - xyz=‘This is a thistle? Yes, this is a thistle.’;
  - a=‘this_’

b	statement	search for	result	xyz
	count(xyz,a)	this_	1	‘This is a thistle? Yes, this is a thistle.’
‘i’	count(xyz,a,b)	this_,This_	2	‘This is a thistle? Yes, this is a thistle.’
‘t’	count(xyz,a,b)	this	3	‘This is a thistle? Yes, this is a thistle.’
‘it’	count(xyz,a,b)	this,This	4	‘This is a thistle? Yes, this is a thistle.’

Function COUNT, COUNTC (2)

- **COUNTC** function
  - Counts the number of specific characters that either appear or do not appear within a character string that you specify
  - Modifiers: i, t, v

- Example:
  - xyz='Baboons Eat Bananas ';
  - a='ab'

b	statement	search for	result	xyz
	countc(xyz,a)	a,b	6	'Baboons_Eat_Bananas_ '
'v'	countc(xyz,a,b)	^(a,b)	14	'Baboons_Eat_Bananas_'
'ivt'	countc(xyz,a,b)	^(a,b,A,B)	11	'Baboons_Eat_Bananas_'