

Disclaimer: This material is protected under copyright act AnalytixLabs ©, 2011. Unauthorized use and/ or duplication of this material or any part of this material including data, in any form without explicit and written permission from AnalytixLabs is strictly prohibited. Any violation of this copyright will attract legal actions.

***Learn to Evolve***

# General Form of SAS Functions

---

- To use a SAS function, specify the function name followed by the function arguments, which are enclosed in parentheses
- Even if the function does not require arguments, the function name must still be followed by parentheses
- Unless the length of the target variable has been previously defined, a **default length** is assigned

## Syntax:

***function-name (argument-1 , <argument-n>);***

where,

- ***arguments*** can be
  - Variables *x,y,z*
  - Constants *456,502,612,498*
  - Expressions *37\*2,192/5, mean(22,34,56)*

## Example:

---

- A function that contains multiple arguments

**std**(x1,x2,x3) ;

**mean** (of x1-x3) ;

AvgScore = **sum** (exam1,exam2,exam3) ;

---

# Arithmetic Functions

# Sum Function

---

- Calculates the sum of values

## Syntax:

**sum**( *argument* , *argument*,...)

where,

- **argument** can be sas variables, constants and expressions

## Example:

---

```
Data work.after;  
Set work.before;  
    totalsal = sum (sal1,sal2,sal3);  
    Run;
```

Here,

- The above program calculates the sum of the values in sal1, sal2 and sal3 variables.

# MEAN Function

---

- Calculates the average of non missing values

## Syntax:

**mean** (*argument, argument,...*)

where,

- **argument** can be sas variables, constants and expressions

## Example:

---

```
Data work.after;  
Set work.before;  
    avg = mean (marks1,marks2,marks3);  
    Run;
```

Here,

- The above program calculates the average of the values in marks1, marks2 and marks3 variables.



# MIN Function

---

- Finds the minimum value

## Syntax:

**min ( *argument, argument,...* )**

where,

- **argument** can be sas variables, constants and expressions

## Example:

---

```
Data work.after;  
Set work.before;  
minimum =min (marks1,marks2,marks3);  
Run;
```

Here,

- The above program finds the minimum of the values in marks1, marks2 and marks3 variables.

# MAX Function

---

- Finds the maximum value

**Syntax:**

***max(argument, argument,...)***

where,

- **argument** can be sas variables, constants and expressions

## Example:

---

```
Data work.after;  
Set work.before;  
maximum =max (marks1,marks2,marks3);  
Run;
```

Here,

- The above program finds the maximum of the values in marks1, marks2 and marks3 variables.

# STD Function

---

- Calculates the standard deviation of the values

**Syntax:**

***std(argument, argument,...)***

where,

- **argument** can be sas variables, constants and expressions

## Example

---

```
Data work.after;  
Set work.before;  
stdev =std (s1, s2, s3);  
Run;
```

Here,

- The above program calculate the standard deviation of the values in s1, s2 and s3 variables.

# Modifying Numeric Values with Functions

---

## INT Function

- Return the integer portion of a numeric value
- Decimal portion of the INT function argument is discarded

**Syntax:**

**INT (*argument*)**

Where,

- ***argument*** is a numeric variable, constant, or expression.

## Example:

---

```
Data work.after;  
Set work.before;  
  Intamt = INT(amount);  
Run;
```

Here,

- The value of the variable amount is converted to integer and stored in a new variable.



# ROUND Function

---

- Round values to the nearest specified unit
- If a round-off unit is not provided, a default value of 1 is used

## Syntax:

**ROUND ( *argument* , *round-off-unit* );**

Where,

- ***argument*** is a numeric variable, constant, or expression.
- ***round-off-unit*** is numeric and nonnegative.

## Example:

---

```
Data work.after;  
Set work.before;  
    amt = ROUND(amount,.2);  
Run;
```

Here,

- value of the variable amount is rounded to 2 decimal points.

## CEIL and FLOOR Functions

---



- The *CEIL function* returns the smallest integer that is **greater** than or equal to the argument.
- The *FLOOR function* returns the largest integer that is **less** than or equal to the argument.
- If the argument is within 1E-12 of an integer, the function returns that integer.

Syntax:

**CEIL** (*argument*)  
**FLOOR** (*argument*)

Example:

num	ceil (num)	floor (num)
2.75	3	2
-2.75	-2	-3
23.1234	24	23
-23.1234	-23	-24



---

# Date & Time Functions

# Manipulating SAS Date Values with Functions

---

## YEAR Function

- Extracts the year value from a SAS date value

**Syntax:**

**YEAR** (*date*);

Where,

- ***date*** is a SAS date value that is specified either as a variable or as a SAS date constant

## Example

---

```
Data hrd.temp98;  
Set hrd.temp;  
           yr = year(startdate);  
Run;
```

Here,

- Year function extracts the year portion from the date value variable startdate and save it in the new variable yr.

# QTR Function

---

- Extracts the quarter value from a SAS date value

## Syntax:

**QTR** (*date*) ;

Where,

- ***date*** is a SAS date value that is specified either as a variable or as a SAS date constant.

## Example

---

```
Data hrd.temp98;  
Set hrd.temp;  
               quarter = qtr(startdate);  
Run;
```

Here,

- QTR function extracts the quarter value from the date value variable startdate and save it in the new variable quarter.



# MONTH Function

---

- Extracts the month value from a SAS date value

## Syntax:

**MONTH** (*date*) ;

where,

- ***date*** is a SAS date value that is specified either as a variable or as a SAS date constant.

## Example

---

```
data hrd.nov99;  
set hrd.temp;  
                mn = month(startdate);  
Run;
```

Here,

- Month function extracts the month value from the startdate variable and save it in the new variable mn.

# DAY Function

---

- Extracts the day value from a SAS date value.

## Syntax:

**DAY** (*date*);

Where,

- ***date*** is a SAS date value that is specified either as a variable or as a SAS date constant

## Example:

---

```
data hrd.nov99;  
set hrd.temp;  
                days = day(date);  
Run;
```

Here,

- Day function extracts the day value from the date variable and save it in the new variable days.

# WEEKDAY Function

---

- Extract the day of the week from a SAS date value

## Syntax:

**WEEKDAY (*date*) ;**

where,

- ***date*** is a SAS date value that is specified either as a variable or as a SAS date constant

## Example

---

```
data hrd.nov99;  
set hrd.temp;  
                weekday = weekday(date);  
Run;
```

Here,

- WEEKDAY function extracts the day of the week value from the date variable and save it in the new variable weekday.

- 
- The WEEKDAY function returns a numeric value from 1 to 7. The values represent the days of the week.

Value	equals	Day of the Week
1	=	Sunday
2	=	Monday
3	=	Tuesday
4	=	Wednesday
5	=	Thursday
6	=	Friday
7	=	Saturday

# MDY Function

---

- Creates a SAS date value from numeric values that represent the month, day, and year

## Syntax:

**MDY ( *month* , *day* , *year* );**

Where,

- ***month*** can be a variable that represents the month, or a number from 1-12
- ***day*** can be a variable that represents the day, or a number from 1-31
- ***year*** can be a variable that represents the year, or a number that has 2 or 4 digits.



## Example:

---

```
data hrd.newtemp (drop=month day year);  
    set hrd.temp;  
    Date= mdy(month,day,year);  
run;
```

Here,

- A new variable date will be created by combining the values in the variables month, day and year using the mdy function.

# DATE and TODAY Functions

---

- Return the current date from the system clock as a SAS date value

## Syntax:

**DATE()**

**TODAY()**

- These functions require no arguments, but they must still be followed by parentheses.

## Example

---

```
data hrd.newtemp;  
      set hrd.temp;  
      EditDate = date();  
run;
```

Here,

- Date function returns the current system date and store it in a new variable editdate.

# TIME Function

---

- Return the current time as a SAS time

## Syntax:

**time ( );**

- This function require no arguments, but it must still be followed by parentheses

## Example:

---

```
data hrd.newtemp;  
      set hrd.temp;  
      starttime = time();  
run;
```

Here,

- TIME function returns the current system time and store it in a new variable starttime.

# INTCK Function

---

- Returns the number of time intervals that occur in a given time span
- Used to count the passage of days, weeks, months, and so on
- Counts intervals from fixed interval beginnings, not in multiples of an interval unit from the *from* value
- Partial intervals are not counted
- For example :
  - WEEK intervals are counted by Sundays rather than seven-day multiples from the ***from*** argument
  - MONTH intervals are counted by day 1 of each month
  - YEAR intervals are counted from 01JAN, not in 365-day multiples

## Syntax:

---

**INTCK (*'interval '*, *from* , *to* );**

Where,

- ***'interval'*** specifies a character constant or variable. The value must be one of the following in the box:
- ***from*** specifies a SAS date, time, or datetime value that identifies the beginning of the time span
- ***to*** specifies a SAS date, time, or datetime value that identifies the end of the time span
- The type of interval (date, time, or datetime) must match the type of value in *from*

DAY	DTMONTH
WEEKDAY	DTWEEK
WEEK	HOUR
TENDAY	MINUTE
SEMIMONTH	SECOND
MONTH	
QTR	
SEMIYEAR	
YEAR	

## Example:

---

```
Data work.anniv20;  
SET flights.mechanics ( KEEP=id lastname firstname hired);  
Years= INTCK ( 'year' , hired , today() );  
If years=20 and Month (hired) = Month (TODAY());  
Proc Print Data = work.anniv20;  
Run;
```

Here,

- The program identifies mechanics whose 20th year of employment occurs in the current month
- It uses the INTCK function to compare the value of the variable Hired to the date on which the program is run.



# INTNX Function

---

- Applies multiples of a given interval to a date, time, or datetime value and returns the resulting value
- Used to identify past or future days, weeks, months, and so on

## Syntax:

**INTNX ( ' *interval* ' , *start-from* , *increment*< , ' *alignment* ' > )**

Where,

- '**interval**' specifies a character constant or variable
- **start-from** specifies a starting SAS date, time, or datetime value
- **increment** specifies a negative or positive integer that represents time intervals toward the past or future

- 
- **'alignment'** (optional) forces the alignment of the returned date to the beginning, middle, or end of the interval.
  - The type of interval (date, time, or datetime) must match the type of value in start-from and increment.
  - When specifying date intervals, the value of the character constant or variable that is used in interval must be one of the following in the box:
  - Optional *alignment* argument lets us specify whether the date value should be at the beginning, middle, or end of the interval.
  - When specifying date alignment in the INTNX function, use the following arguments or their corresponding aliases:

- BEGINNING      B
- MIDDLE         M
- END             E
- SAME            S

DAY	DTMONTH
WEEKDAY	DTWEEK
WEEK	HOUR
TENDAY	MINUTE
SEMIMONTH	SECOND
MONTH	
QTRSEMIYEAR	
YEAR	

## Example:

---

<b>SAS Statement</b>	<b>Date Value</b>
MonthX = intnx ('month','10jan95'd,5,'b');	12935 (June 1, 1995)
MonthX = intnx ('month','10jan95'd,5,'m');	12949 (June 15, 1995)
MonthX = intnx ('month','10jan95'd,5,'e');	12964 (June 30, 1995)
MonthX = intnx ('month','10jan95'd,5,'s');	12944 (June 10, 1995)

- The statements above count five months from January, but the returned value depends on whether alignment specifies the beginning, middle, or end day of the resulting month.
- If alignment is not specified, the beginning day is returned by default.

# DATEPART Function

---

- To separate the date portion from date and time value

**Syntax:**

**Datepart** (*variable*);

where,

- ***variable*** specifies the name of the variable

## Example

---

```
data hrd.newtemp;  
set hrd.temp;  
          Date = datepart(saledate);  
run;
```

Here,

- Datepart function extracts the date portion from saledate, which is in date and time format, and save it in new variable date .

# DATDIF Functions

---

- Calculate the difference in days between two SAS dates
- Accept dates that are specified as SAS date values

## Syntax:

**DATDIF( *start\_date* , *end\_date* , *basis* ) ;**

Where,

- ***start\_date*** specifies the starting date as a SAS date value
- ***end\_date*** specifies the ending date as a SAS date value
- ***basis*** specifies a character constant or variable that describes how SAS calculates the date difference.

## Example

---

```
data hrd.newtemp;  
    set hrd.temp;  
    date= DATDIF(sdate,edate,'ACT/ACT');  
run;
```

Here,

- DATDIF function gives the difference between two dates in number of days.

# YRDIF Function

---

- Calculate the difference in years between two SAS dates
- Accept start dates and end dates that are specified as SAS date values
- Use a **basis** argument that describes how SAS calculates the date difference

## Syntax

**YRDIF** ( *start\_date* , *end\_date* , '*basis*' )

where,

- **start\_date** specifies the starting date as a SAS date value
- **end\_date** specifies the ending date as a SAS date value
- **basis** specifies a character constant or variable that describes how SAS calculates the date difference.



---

### Example:

```
data hrd.newtemp;  
    set hrd.temp;  
    date= YRDIF (sdate, edate, 'ACT/ACT');  
run;
```

Here,

- YRDIF function gives the difference between the two dates in number of years.

---

There are two character strings that are valid for basis in the DATDIF function and four character strings that are valid for basis in the YRDIF function. These character strings and their meanings are listed in the table below.

Character String	Meaning	Valid In DATDIF	Valid In YRDIF
' 30 / 360 '	specifies a 30 day month and a 360 day year	yes	yes
' ACT / ACT '	uses the actual number of days or years between dates	yes	yes
' ACT / 360 '	uses the actual number of days between dates in calculating the number of years (calculated by the number of days divided by 360)	no	yes
' ACT / 365 '	uses the actual number of days between dates in calculating the number of years (calculated by the number of days divided by 365)	no	yes

---

# Text Functions

# Converting Data with Functions

---

## INPUT function

Explicitly convert the character values to numeric values

**Syntax:**

**INPUT** (*source*, *informat* );

**Where.**

- ***source*** indicates the character variable, constant, or expression to be converted to a numeric value
- ***informat*** is the numeric *informat* to be specified. When choosing the informat, be sure to select a numeric informat that can read the form of the values.

## Example

---

```
Data hrd.newtemp;  
Set hrd.temp;  
    Test=input(saletest,comma9.);  
Run;
```

Here,

The function uses the numeric informat COMMA9. to read the values of the character variable SaleTest. Then the resulting numeric values are stored in the variable Test.

Character Value	Informat
2115233	7.
2,115,233	COMMA9.

# PUT Function

---

- Explicitly convert the numeric values to character values
- Format specified in the PUT function must match the data type of the source

## Syntax:

**PUT(*source,format*) ;**

Where,

- ***source*** indicates the numeric variable, constant, or expression to be converted to a character value
- ***format*** specifies the matching data type of the source

- 
- The PUT function always returns a character string.
  - The PUT function returns the *source* written with a *format*.
  - The *format* must agree with the *source* in type.
  - Numeric formats right-align the result; character formats left-align the result.
  - If you use the PUT function to create a variable that has not been previously identified, it creates a character variable whose length is equal to the format width.

## Example

---

```
data hrd.newtemp;  
set hrd.temp;  
Assignment = put (site,2.) || '/' || dept;  
run;
```

Here,

- Because Site has a length of 2, its given 2. as the numeric format.
- Put function converts the data type of site variable into character data type.
- After that the value is concatenated and saved in the new variable assignment.



# Modifying Character Values with Functions

---

## SCAN Function:

- Enables you to separate a character value into words and to return a specified word
- Uses **delimiters**, which are characters that are specified as word separators, to separate a character string into words
- Can specify as many delimiters as needed to correctly separate the character expression
- The default delimiters are

blank . < ( + | & ! \$ \* ) ; ^ - / , %

## Syntax:

---

**SCAN (*argument* , *n* , *delimiters*);**

where,

- ***argument*** specifies the character variable or expression to scan
- ***n*** specifies which word to read
- ***delimiters*** are special characters that must be enclosed in single quotation marks ( ' ' ).

## Example:

---

```
Data hrd.newtemp ( DROP=name);  
Set hrd.temp;  
    LastName = SCAN (name ,1 , ' ');  
    FirstName =SCAN (name , 2 , ' ');  
    MiddleName =SCAN (name ,3 , ' ');  
Run;
```

Here,

- It creates three variables to store the employee's first name, middle name & last name which is stored in a variable called name

# SUBSTR Function:

---

- Extract a portion of a character value
- Replace the contents of a character value
- When the function is on the right side of an assignment statement, the function returns the requested string
- When the function is on the left side of an assignment statement, the function is used to modify variable values

## Syntax:

**SUBSTR (*argument*, *position*, <*n*>)**

Where,

- ***argument*** specifies the character variable or expression from which to extract substring.
- ***position*** is the character position to start from.
- ***n*** specifies the number of characters to extract. If *n* is omitted, all remaining characters are included in the substring.

## Example:

---

```
Data work.newtemp (DROP = middlename);  
Set hrd.newtemp;  
    MiddleInitial = Substr ( middlename , 1 ,1 );  
Run;
```

Here,

- It extract the first letter of the MiddleName value to create the new variable MiddleInitial.

```
Data hrd.temp2 (DROP = exchange );  
Set hrd.temp;  
    Exchange= Substr ( phone , 1 , 3 );  
    If exchange='622' Then Substr (phone , 1 , 3) = '433';  
Run;
```

Here,

It searches the value 622 and replace with 433 in the variable phone

## SCAN Function Compared with SUBSTR Function:

---

- **SCAN** extracts words within a value that is marked by delimiters
- The SCAN function is best used when we
  - know the order of the words in the character value
  - the starting position of the words varies
  - the words are marked by some delimiter
- **SUBSTR** extracts a portion of a value by starting at a specified location
- SUBSTR function is best used when the exact position of the substring that is to be extracted from the character value is known
- Substring does not need to be marked by delimiters

# TRIM Function:

---

- Enables to remove trailing blanks from character values
- Whenever the value of a character variable does not match the length of the variable, SAS pads the value with trailing blanks
- So problem occurs while concatenating two variable values.
- Trim the values of a variable and then assign these values to a new variable, the trimmed values are padded with trailing blanks again if the values are shorter than the length of the new variable

## Syntax:

**TRIM ( *argument* )**

Where,

- ***argument*** can be any character expression, such as
- a character variable: trim ( address )
- another character function: trim (left (id) )

## Example:

---

```
Data hrd.newtemp ( Drop = address city state zip);  
Set hrd.temp;  
    NewAddress = Trim (address) || ', ' || TRIM (city) || ', ' || zip;  
Run;
```

Here,

- A new variable called **newaddress** is created which contain the full address taken from three different variables called address, city and zip
- The trailing spaces of the variables address and city are trimmed using trim function .



## CAT Functions:

---

- Enables to concatenate character strings, remove leading and trailing blanks, and insert separators
- Returns a value to a variable or returns a value to a temporary buffer

CAT	does not remove leading or trailing blanks.
	<code>FullName1 = cat(First, Middle, Last);</code>
CATS	removes leading and trailing blanks.
	<code>FullName2 = cats(First, Middle, Last);</code>
CATT	removes trailing blanks.
	<code>FullName3 = catt(First, Middle, Last);</code>
CATX	removes leading and trailing blanks and inserts separators.
	<code>FullName4 = catx(' ', First, Middle, Last);</code>

Example:

```
data name;  
  length First Middle Last $ 10;  
  First = 'Tony';  
  Middle = ' Albert';  
  Last = 'Smith';  
  Name1 = cat(First, Middle, Last);  
  Name2 = cats(First, Middle, Last);  
  Name3 = catt(First, Middle, Last);  
  Name4 = catx(' ', First, Middle, Last);  
run;
```

First (10 bytes)	Middle (10 bytes)	Last (10 bytes)
Tony	Albert	Smith

New Variables (200 bytes)		
Name1	Tony     Albert   Smith	CAT does not remove blanks.
Name2	TonyAlbertSmith	CATS removes leading and trailing blanks.
Name3	Tony   AlbertSmith	CATT removes trailing blanks.
Name4	Tony Albert Smith	CATX is CATS plus adds separators.

## CATX Function:

---

- Enables to concatenate character strings, remove leading and trailing blanks, and insert separators
- Returns a value to a variable, or returns a value to a temporary buffer
- Results of the CATX function are usually equivalent to those that are produced by a combination of the concatenation operator and the TRIM and LEFT functions

### Syntax:

**CATX ( *separator* , *string-1* <,...*string-n*> )**

Where,

- ***separator*** specifies the character string that is used as a separator between concatenated strings
- ***string*** specifies a SAS character string.

## Example:

---

```
Data hrd.newtemp ( DROP = address city state zip);  
Set hrd.temp;  
      NewAddress = CATX ( ' , ' , address , city , zip);  
Run;
```

Here,

- The above program uses CATX function to concatenate the variables address, city & zip into new variable newaddress and separates each values with comma.

# INDEX Function:

---

- Enables to search a character value for a specified string
- Searches values from left to right, looking for the first occurrence of the string
- Returns the position of the string's first character
- If the string is not found, it returns a value of *0*
- Is case sensitive

## Syntax:

**INDEX (*source* ,*excerpt* )**

Where,

- ***source*** specifies the character variable or expression to search
- ***excerpt*** specifies a character string that is enclosed in quotation marks ( ' ' ).

## Example:

---

```
Data hrd.datapool;  
Set hrd.temp;  
    If Index ( job , 'word processing' ) > 0;  
Run;
```

Here,

- It is creating a new dataset with only those observations, in which the function locates the string 'word processing' and returns a value greater than 0.

## FIND Function:

---

- Search for a specific substring of characters within a character string specified
- Returns the position of that substring
- If the substring is not found in the string, returns a value of 0
- Similar to the INDEX function

## Syntax:

---

**FIND** (string , substring , <modifiers> , < startpos> )

Where,

- **string** specifies a character constant, variable, or expression that will be searched for substrings
- **substring** is a character constant, variable, or expression that specifies the substring of characters to search for in **string**
- **startpos** is an integer that specifies the position at which the search should start and the direction of the search
  - If startpos is not specified, FIND starts the search at the beginning of the string and searches the string from left to right.
  - If startpos is positive, FIND searches from startpos to the right
  - If startpos is negative, FIND searches from startpos to the left
- **modifiers** argument specifies one or more modifiers for the function, as listed below.
  - The modifier **i** causes the FIND function to ignore character case during the search. If this modifier is not specified, FIND searches for character substrings with the same case as the characters in substring.
  - The modifier **t** trims trailing blanks from **string** and **substring**



## Example:

---

```
Data hrd.datapool;  
Set hrd.temp;  
  If Find ( job , ' word processing ' , ' t ' ) > 0;  
Run;
```

Here,

- It Creates a new dataset with only those observations, in which the function locates the string 'word processing' and returns a value greater than 0.

## Difference between Index and Find function

---

The INDEX function and the FIND function both search for substrings of characters in a character string. However, the FIND function can have modifiers and specify a starting position.

## UPCASE Function:

---

- Converts all letters in a character expression to uppercase

### Syntax:

**UPCASE (*argument*)**

Where,

- ***argument*** can be any SAS expression, such as a character variable or constant

## Example:

---

```
Data hrd.newtemp;  
Set hrd.temp;  
    Job = UPCASE (job) ;  
Run;
```

Here,

- The above program converts the values of Job to uppercase and save into a new dataset.

## LOWCASE Function:

---

- Converts all letters in a character expression to lowercase

### Syntax:

**LOWCASE ( *argument* )**

Where,

- ***argument*** can be any SAS expression, such as a character variable or constant.

## Example:

---

```
Data hrd.newtemp;  
Set hrd.temp;  
    Contact = LOWCASE ( contact);  
Run;
```

Here,

- The above program converts the values of variable contact to lowercase and store in a new dataset.

## PROPCASE Function:

---

- Converts all words in an argument to proper case (the first letter in each word is capitalized)
- First copies a character argument and converts all uppercase letters to lowercase letters
- Then converts to uppercase the first character of a word that is preceded by a delimiter
- Uses the default delimiters unless specified

### Syntax:

**PROPCASE (argument , <delimiter (s)> )**

Where,

- **argument** can be any SAS expression, such as a character variable or constant
- **delimiter(s)** specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

## Example:

---

```
Data hrd.newtemp;  
Set hrd.temp;  
  Contact = PROPCASE(contact);  
Run;
```

Here,

- The program converts the values of variable contact into proper case and save into new dataset.



## TRANWRD Function :

---

- Replaces or removes all occurrences of a pattern of characters within a character string
- Translated characters can be located anywhere in the string

### Syntax

**TRANWRD** (*source, target, replacement*)

where

- ***source*** specifies the source string that you want to translate
- ***target*** specifies the string that SAS searches for in *source*
- ***replacement*** specifies the string that replaces *target*.
- ***target*** and ***replacement*** can be specified as variables or as character strings

---

### Example:

```
Data work.after;  
Set work.before;  
  name = TRANWRD (name, 'Miss', 'Ms.');
```

```
      name = TRANWRD (name, 'Mrs. ', 'Ms.');
```

```
Run;
```

Here,

- The above program change all occurrences of *Miss* or *Mrs.* to *Ms.* in the variable name.

# Translate Function

---

- Replaces or removes all occurrences of a character within a character string

## Syntax

**TRANSLATE(source, < to 1-n>, < from 1-n>)**

where,

- **source** specifies the source string or name of the variable whose value is to be translated
- **to 1-n** specifies the characters to be replaced with
- **from 1-n** specifies the characters to be replaced

## Example:

---

```
Data work.after;  
Set work.before;  
name = TRANSLATE (name, 'XYZ', 'ABC.');
```

**Run;**

Here,

- The above program will replace all the A's with X, B's with Y and C's with Z in the name variable.

# Left and Right Functions

---

- The *LEFT function* left-aligns a character expression.

LEFT returns an argument with leading blanks moved to the end of the value.

- The *RIGHT function* right-aligns a character expression.

RIGHT returns an argument with trailing blanks moved to the start of the value.

## Syntax:

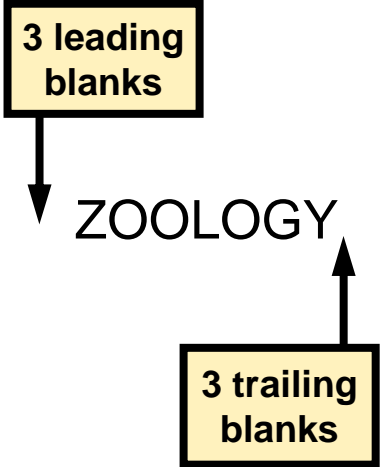
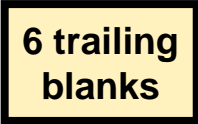
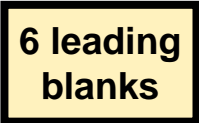
**LEFT ( *source* );**  
**RIGHT ( *source* );**

Where,

***source*** specifies the source string or name of the variable.

If the LEFT or RIGHT function returns a value to a variable that was not yet assigned a length, the variable length is determined by the length of the argument.

## Example:

VAR (13 bytes)	Assignment Statement Using the LEFT or RIGHT Function	New Variable (13 bytes)
	<code>NewVar1 = left(Var);</code>	ZOOLOGY 
	<code>NewVar2 = right(Var);</code>	 ZOOLOGY

# Concatenation Operator

---

The *concatenation operator* (|| - two vertical bars, || - two broken vertical bars, or !! - two exclamation points) concatenates character values.

## Syntax:

Variable = Variable1 || Variable 2 || Variable 3;

The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation, unless you use a LENGTH statement to specify a different length for the new variable.

The concatenation operator does not trim leading or trailing blanks. If variables are padded with trailing blanks, use the TRIM function to trim trailing blanks from values before concatenating them.

## Example:

---

```
Data name;  
length Name $ 20 First Middle Last $ 10;  
Name = 'Jones, Mary Ann, Sue';  
First = left(scan(Name,2,','));  
Middle = left(scan(Name,3,','));  
Last = scan(name,1,',');  
FullName = First || Middle || Last;  
run;
```

First (10 bytes)	Middle (10 bytes)	Last (10 bytes)
Mary Ann	Sue	Jones

FullName (30 bytes)		
Mary	Ann	Sue
Jones		

2 blanks      7 blanks      5 blanks



# SAS System Options

---

- Are used to modify system options
- Can place an OPTIONS statement anywhere in a SAS program to change the settings from that point onwards
- OPTIONS statement is **global** ie: the settings remain in effect until modify them, or end SAS session

## Syntax:

**OPTIONS** *options*;

Where,

***options*** specifies one or more system options to be changed

- The available system options depend on the host operating system

# NUMBER | NONNUMBER and DATE | NODATE Options:

---

- Page numbers and dates appear with output

## NONNUMBER & NODATE Options:

### Syntax:

**options** nonnumber nodate;

- This suppresses the printing of both page numbers and the date and time in listing output

## NUMBER & DATE Options:

### Syntax:

**options** nonnumber nodate;

- This prints both page numbers and the date & time in listing output

## Example:

---

```
options nonumber nodate;  
proc print data=clinic.admit ;  
  var id sex age height weight;  
  where age>=30;  
run;  
options date;  
proc freq data = clinic.diabetes;  
  where fastgluc >= 300;  
  tables sex;  
run;
```

Here,

- Page numbers and the current date are not displayed in the PROC PRINT output
- Page numbers are not displayed in the PROC FREQ output, either, but the date does appear at the top of the page that contains the PROC FREQ report

## Output:

---

The SAS System					
Obs	ID	Sex	Age	Height	Weight
2	2462	F	34	66	152
3	2501	F	31	61	123
4	2523	F	43	63	137
5	2539	M	51	71	158
7	2552	F	32	67	151
8	2555	M	35	70	173

The SAS System				
15:19 Thursday, September 23, 1999				
Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	2	25.0	2	25.0
M	6	75.0	8	100.0

## PAGENO, PAGESIZE & LINESIZE Options:

---

- PAGENO= option is used to specify the beginning page number for the report
- If its not specified, the output is numbered sequentially throughout the SAS session, starting with page 1
- The PAGESIZE= option specifies how many lines each page of output should contain
- The LINESIZE= option specifies the width of the print line for the procedure output and log
- Observations that do not fit within the line size continue on a different line

### Syntax:

**options pageno = n pagesize =n linesize = n;**

Where,

**n** is any number

## Example:

---

```
options pageno =1 pagesize=15 linesize =64 ;  
proc print data = clinic.admit ;  
run ;
```

Here,

- The output pages are numbered sequentially throughout the SAS session
- The page of the output that the PRINT procedure produces contains 15 lines
- The length of the observations are no longer than 64 characters

## YEARCUTOFF Option:

---

- This option specifies which 100-year span is used to interpret two-digit year values
- When a two-digit year value is read, SAS interprets it based on a 100-year span that starts with the YEARCUTOFF= value
- The default value of YEARCUTOFF= is **1920**
- The default value of yearcutoff can be changed using the YEARCUTOFF= option
- The value of the YEARCUTOFF= system option affects only **two-digit year values**

1920 ← 100 years → 2019

Date Expression	Interpreted As
12/07/41	12/07/1941
18Dec15	18Dec2015
04/15/30	04/15/1930
15Apr95	15Apr1995



**Syntax:**

---

**options YEARCUTOFF = YEAR;**

Where,

YEAR is the first year of the 100 year span



## Example:

---

**options** yearcutoff =1950 ;

Here,

- The 100-year span will be from 1950 to 2049
- Using YEARCUTOFF=1950, dates are interpreted as shown below:

**1950 ← 100 years → 2049**

Date Expression	Interpreted As
12/07/41	12/07/2041
18Dec15	18Dec2015
04/15/30	04/15/2030
15Apr95	15Apr1995

## OBS, FIRSTOBS options:

---

- Used to specify the observations to process from SAS data sets
- Can specify either or both of these options as needed
- OBS= to specify the last observation to be processed
- FIRSTOBS= to specify the first observation to be processed
- FIRSTOBS= and OBS= together to specify a range of observations to be processed

## Syntax:

---

**OPTIONS FIRSTOBS= $n$ ;**

**OPTIONS OBS= $n$ ;**

Where,

$n$  is a positive integer

For FIRSTOBS=,  $n$  specifies the number of the **first** observation to process

For OBS=,  $n$  specifies the number of the **last** observation to process

By default, FIRSTOBS=1. The default value for OBS= is MAX

## Example:

---

```
options firstobs =10 ;  
proc print data =sasuser.heart ;  
run ;
```

- Assume the data set Sasuser.Heart contains 20 observations.
- Here SAS reads the 10th observation of the data set first and reads through the last observation (for a total of 11 observations)

```
options firstobs =1 obs =10 ;  
proc print data =sasuser.heart ;  
run ;
```

- Here SAS reads 1<sup>st</sup> to 10th observation (for a total of 10 observations)

- 
- To reset the number of the last observation to process, you can specify OBS=MAX in the OPTIONS statement.

**options obs = max;**

- This instructs any subsequent SAS programs in the SAS session to process through the last observation in the data set being read
- Obs and firstobs will be for the duration of current SAS session

## Viewing System Options:

---

- OPTIONS procedure can be used to display the current setting of one or all SAS system options
- The results are displayed in the log

### Syntax:

```
PROC OPTIONS < option (s) > ;  
RUN;
```

Where, *option(s)* specifies how SAS system options are displayed

### Example:

```
proc options;  
Run;
```

- This lists all SAS system options, their settings, and a description

- 
- To list the value of one particular system option, use the `OPTION=` option in the `PROC OPTIONS` statement as shown below:

```
proc options option = yearcutoff ;  
run ;
```

- If a SAS system option uses an equal sign, such as `YEARCUTOFF=`, you do not include the equal sign when specifying the option to `OPTION=`.