**SAS Date and Time Functions:**

These SAS functions are used to perform operations on date and time values.

|  |  |
| --- | --- |
| DATE() | returns the current date as a SAS date value |
| DATETIME() | returns the current date and time of day |
| DAY(date) | returns the day of the month from a SAS date value |
| INTCK(‘interval’,from,to) | returns the number of time intervals in a given time span |
| MONTH(date) | returns the month from a SAS date value |
| QTR(date) | returns the quarter of the year from a SAS date value |
| TIME() | returns the current time of day |
| TODAY() | returns the current date as a SAS date value |
| WEEKDAY(date) | returns the day of the week from a SAS date value |
| YEAR(date) | returns the year from a SAS date value |
| YRDIF(sdate,edate,basis) | returns the difference in years between two dates |
|

**Example:**

data date\_functions;

INPUT @1 date1 date9. @11 date2 date9.;

format date1 date9. date2 date9.;

/\* Get the interval between the dates in years\*/

Years\_ = INTCK('YEAR',date1,date2);

/\* Get the interval between the dates in months\*/

months\_ = INTCK('MONTH',date1,date2);

/\* Get the week day from the date\*/

weekday\_ = WEEKDAY(date1);

/\* Get Today's date in SAS date format \*/

today\_ = TODAY();

/\* Get current time in SAS time format \*/

time\_ = time();

DATALINES;

21OCT2000 16AUG1998

01MAR2009 11JUL2012;

run;

**INTCK:** This is used to calculate the difference between two dates, two times or two date time values.

**INTCK(date-or-time-interval, start-date-or-time, end-date-or-time, [method])**

1. date-or-time-interval: Date or time period needs to be defined in the first parameter. For eg. MONTH, YEAR, QTR, WEEK, HOUR, MINUTE etc. Specify period in single quotes

2. start-date-or-time: Starting date or time to calculate the number of periods.

3. end-date-or-time: End date or time to calculate the number of periods.

4. Method: Optional Parameter. Method to calculate the difference. Methods are 'CONTINUOUS' or 'DISCRETE'. By default, it is DISCRETE.

data temp;  
date1 = '01JAN2015'd;  
date2 = '01JAN2017'd;  
no\_of\_years  = intck ('YEAR', date1, date2);  
format date1 date2 date9.;  
proc print data = temp;  
run;

**INTNX :** INTNX is used to increment SAS date by a specified number of intervals

**INTNX(interval, start-from, increment,  [alignment])**

1.Interval is the unit of measurement. The intervals can be days, weeks, months, quarters, years.

Start-from is a SAS date value which would be incremented.

2. Increment is number of intervals by which date is incremented. It can be zero, positive or negative. Negative value refers to previous dates.

3. Alignment [Optional Parameter] is where date value is aligned within interval prior to being incremented. The values you can specify - 'beginning', 'middle', 'end', 'same day'. Default value - 'beginning'.

data temp;  
mydate = '02JAN2017'd;  
day=intnx('day', mydate , 7);  
format mydate day date9.;  
run;

Result : day = 09JAN2017

**CONVERTING NUMBER FORMAT TO DATE FORMAT:**

**PUT**: Function is used to convert the numeric variable to character format

**INPUT**: Function is used to convert the character variable to sas date format

data out;  
set example;  
dateofbirth2 = input(strip(dateofbirth),MMDDYY10.);  
format dateofbirth2 MMDDYY10.;  
run;

**FORMAT:**Function is used to display the SAS date values in a particular SAS date format. If we would not use format function, SAS would display the date in SAS date values format. For example, 20588 is a sas date value and it is equivalent to '14MAY2016'.

**Compress Function:**

1. The COMPRESS function compresses the character value and removes all of the blank spaces from the string.

data string2;  
set string;  
comp = compress(text);  
run;

1. We can also specify a character to be removed in the COMPRESS function

data string2;  
set string;  
comp = compress(text, 'a');  
run;

The character 'a' is specified in the second parameter of the COMPRESS function. It will be removed from the text

1. Multiple characters can be removed simultaneously in the COMPRESS function.

data string2;  
set string;  
comp = compress(text, 'abc');  
run;

The 3 letters 'a', 'b', and 'c' are specified in the second parameters. ​All 3 letters will be removed from the text.

1. Adding the 'i' modifier to the third parameter of the COMPRESS function tells SAS to ignore the case of the characters when removing them from the text.

data string2;  
set string;  
comp = compress(text, 'abc', 'i');  
run;

1. We can use the 'a' modifier to remove all of the alphabetic characters from the text.

data string2;  
set string;  
comp = compress(text, '', 'a');  
run;

In addition to the blank space specified in the second parameter, the 'a' modifier in the third parameter tells SAS to remove all of the alphabetic characters from the text.

1. Similar to the 'a' modifier, the 'd' modifier removes all of the digits from the text.

data string2;  
set string;  
comp = compress(text, '', 'd');  
run;

**Numeric Function:**

|  |  |
| --- | --- |
| Common Functions | Example |
| INT: the integer portion of a numeric value | a = int(x); |
| ABS: the absolute value of the argument | a = abs(x); |
| SQRT: the square root of the argument | a = sqrt(x); |
| MIN: the minimum value of the arguments | a = min(x, y, z); |
| MAX: the maximum value of the arguments | a = max(x, y, z); |
| SUM: the sum of the arguments | a = sum(x, y, z); |
| MEAN: the mean of the arguments | a = mean(x, y, z); |
| ROUND: round the argument to the specified unit | a = round(x, 1); |
| LOG: the log (base e) of the argument | a = log(x); |
| LAG: the value of the argument in the previous observation | a = lag(x); |
| DIF: the difference between the values of the argument in the current and previous observations | a = dif(x); |
| N: the number of non-missing values of the argument | a = n(x); |
| NMISS: the number of missing values of the argument | a = nmiss(x); |

**Example:**

data Math\_functions;

v1=21; v2=42; v3=13; v4=10; v5=29;

/\* Get Maximum value \*/

max\_val = MAX(v1,v2,v3,v4,v5);

/\* Get Minimum value \*/

min\_val = MIN (v1,v2,v3,v4,v5);

/\* Get Median value \*/

med\_val = MEDIAN (v1,v2,v3,v4,v5);

/\* Get a random number \*/

rand\_val = RANUNI(0);

/\* Get Square root of sum of the values \*/

SR\_val= SQRT(sum(v1,v2,v3,v4,v5));

run;

**Truncation Functions:**

These SAS functions returns integer values by rounding up the data values.

|  |  |
| --- | --- |
| CEIL(argument) | returns the smallest integer that is greater than or equal to the argument |
| FLOOR(argument) | returns the largest integer that is less than or equal to the argument |
| FUZZ(argument) | returns the nearest integer if the argument is within 1E-12 |
| INT(argument) | returns the integer value |
| ROUND(argument,round-off-unit) | rounds to the nearest round-off unit |
| TRUNC(number, length) | truncates a numeric value to a specified length |

**Example:**

data trunc\_functions;

/\* Nearest greatest integer \*/

ceil\_ = CEIL(11.85);

/\* Nearest greatest integer \*/

floor\_ = FLOOR(11.85);

/\* Integer portion of a number\*/

int\_ = INT(32.41);

/\* Round off to nearest value \*/

round\_ = ROUND(5621.78);

run;

**SUBSTR Function:**

SAS String Functions – SUBSTR Function

**Purpose:** This function extracts a part of a string.

**Syntax:**  SUBSTR(string, start, length )

A Start is the starting position from where we want the string.

length is the number of characters to include in the substring.

If this argument omits, the SUBSTR function will return all the characters from the start position to the end of the string.

**Examples:** let STRING = “ABCXYZ”

i. SUBSTR(STRING,4,2 ) will return “XY”

ii. SUBSTR(STRING,4) will return  “XYZ”

**SAS LOWCASE, UPCASE, and PROPCASE:**

**The SAS LOWCASE Function –** Converts the character string into lowercase character.

**SAS UPCASE Function –** Converts the character string into uppercase character

**SAS PROPCASE Function –** Returns the word having uppercase in the first letter and lowercase in the rest of the letter (sentence format).

|  |  |  |
| --- | --- | --- |
| **Proc** | **Syntax** | **Description** |
| Proc import | PROC IMPORT  DATAFILE=<'filename'>|DATATABLE=<'tablename'>  <DBMS>=<data-source-identifier>  <OUT>=<libref.SAS data-set-name> <SAS data-set-option(s)>  <REPLACE>; | The IMPORT procedure reads external data and writes the data to a SAS data set. |
| Proc export | PROC EXPORT  DATA=<libref.>SAS data-set <(SAS data-set-options)>  OUTFILE="filename" | OUTTABLE="tablename"  <DBMS=identifier> <LABEL><REPLACE>; | The EXPORT procedure creates external output data set from SAS data set |
| Proc print | Proc print data = dsn;  Var var1 var2 var;  Run;   * NOOBS   suppresses the observation number in the output.  For FIRSTOBS= specifies the number of the first observation to process. For OBS= specifies the number of the last observation to process.  options firstobs=10 obs=15;  proc print data=sasuser.heart;  run; | PROC PRINT tells SAS to execute the print procedure on the dataset specified by the DATA command. |
| Proc printto | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* PROGRAM NAME: SASTEST01.SAS \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  %let sasprgnm=sastest01\_part1;  options source2; /\* puts code in log \*/  **proc printto**  **log = "c:\SAS Programs\SAS Logs\**  **&sasprgnm..log"**  print="c:\SAS Programs\SAS Lists\  &sasprgnm..out" new;  %include "c:\SAS Programs\  &sasprgnm..sas";  proc printto;  run; | Proc Printto is used to save log in the desired location. |
| Proc Sort | Proc sort data=employee;  by id;  run;   * The NODUPKEY option removes duplicate observations where value of a variable listed in BY statement is repeated while NODUP option removes duplicate observations where values in all the variables are repeated (identical observations). * Use the DUPOUT= option with NODUPKEY (or NODUP) to output duplicates to the specified SAS data set:   PROC SORT DATA = readin NODUPKEY DUPOUT= readin1;  BY ID;  RUN; | Orders SAS data set observations by the values of one or more character or numeric variables. |
| Proc Means | Proc Means Data = test;  Var q1 - q5;  Run;   * Can produce percentiles p1, p5, p10, p90, p95 , p99 & first and third quartiles (Q1 & Q3) * Can produce statistical options like VAR, range, STDERR etc. * We can use the CLASS statement to group or classify the analysis.   Proc Means data = test N NMISS NOLABELS;  Class Age;  Var q1 - q5;  Run;   * We use NOPRINT option to tell SAS not to print output in output window.   Proc Means data = test NOPRINT;  Var q1 - q5;  Output out = readin mean= median = /autoname;  Run; | It is mainly used to calculate descriptive statistics such as mean, median, count, sum etc. It can also be used to calculate several other metrics such as percentiles, quartiles, standard deviation, variance and sample t-test. |
| Proc Summary | proc summary data = sashelp.shoes print;  var Returns;  run;  Proc SUMMARY and Proc MEANS are essentially the same procedure. Both procedures compute descriptive statistics. The main difference concerns the default type of output they produce. Proc MEANS by default produces printed output in the LISTING window or other open destination whereas Proc SUMMARY does not. Inclusion of the print option on the Proc SUMMARY statement will output results to the output window | This is used to summarize output data. |
| Proc Univariate | proc univariate data = sashelp.shoes;  var sales;  run  **Calculate Custom Percentiles**  proc univariate data = sashelp.shoes noprint;  var sales;  output out = temp  pctlpts = 97.5,99.5 pctlpre = p\_;  run;   * PCTLPTS option helps to calculate custom percentiles. The OUTPUT OUT= statement is used to tell SAS to save the percentile information in TEMP dataset. The PCTLPRE= is used to add prefix in the variable names for the variable that contains the PCTLPTS= percentile. | It is one of the most powerful SAS for running descriptive statistics as well as checking important assumptions of various statistical techniques such as procedure normality, detecting outliers. |
| Proc transpose | |  |  |  |  | | --- | --- | --- | --- | | Obs | famid | year | faminc | | 1 | 1 | 96 | 40000 | | 2 | 1 | 97 | 40500 | | 3 | 1 | 98 | 41000 | | 4 | 2 | 96 | 45000 | | 5 | 2 | 97 | 45400 | | 6 | 2 | 98 | 45800 | | 7 | 3 | 96 | 75000 | | 8 | 3 | 97 | 76000 | | 9 | 3 | 98 | 77000 |   **proc transpose data=long1 out=wide1 prefix=faminc;**  **by famid ;**  **id year;**  **var faminc;**  **run;**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Obs | famid | \_name\_ | faminc96 | faminc97 | faminc98 | | 1 | 1 | faminc | 40000 | 40500 | 41000 | | 2 | 2 | faminc | 45000 | 45400 | 45800 | | 3 | 3 | faminc | 75000 | 76000 | 77000 | | The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations. |
| Proc Rank | proc rank data= temp groups = 4 out = result;  var Score;  ranks ranking;  run; | Calculate rank for one or more numeric variables |
| Proc GLM | proc glm data=exp;  class A B;  model Y=A B A\*B;  run; | This proc is used for both continuous and categorical variables |
| Proc reg | proc reg;  model y = x1-x10;  run; | This proc is used for linear regression modelling. |
| Proc Logistic | proc logistic data = mydata;  class Rating / param = ref;  model Attrition = Tenure | Rating @2 / selection = stepwise slentry=0.15 slstay=0.20;  run; | This proc is used for logistic regression modelling. |
| Proc ARIMA (ADF) | PROC ARIMA DATA= Masterdata ;  IDENTIFY VAR = log\_Air STATIONARITY= (ADF) ;  RUN; | This proc is used for time series modelling. |
| Proc corr | PROC CORR DATA= example1 PEARSON SPEARMAN VAR weight height ;  RUN; | This proc is used for correlation analysis. |
| Proc SQL | PROC SQL;  CREATE TABLE table-name AS  SELECT column(s)  FROM table(s) | view(s)  WHERE expression  GROUP BY column(s)  HAVING Condition  ORDER BY column(s);  QUIT; | PROC SQL can sort, summarize, subset, join (merge), and concatenate datasets, create new variables, and print the results or create a new table or view all in one step |
|  |  |  |

**SAS MACROS :**

**1.How to define macros ?**

The %**MACRO** statement begins the **definition** of a **macro**. It assigns the **macro** a name, and can include a list of **macro** parameters, a list of options, or both.

A **macro definition** must precede the invocation of that **macro** in your code. The %**MACRO** statement can appear anywhere in a **SAS** program, except within data lines.

|  |
| --- |
| Syntax: |

|  |
| --- |
| **%MACRO**  *macro-name* <(*parameter-list*)></ *option-1 <...option-n>*>; |
| ***macro-name***  It names the macro. A macro name must be a SAS name; we cannot use a text expression to generate a macro name in a %MACRO statement.  ***parameter-list***  names one or more local macro variables whose values we can specify when we invoke the macro. Parameters are local to the macro that defines them.  ***parameter-list*** can be   |  |  | | --- | --- | |  | <***positional parameter-1***><. . . ***,positional parameter-n***> | |  | <***keyword-parameter***=<***value***> <. . . ***,keyword-parameter-n***=<***value***>>> |   **2.How to define a macro variable ?**  Macro variables can be created using a number of different techniques and statements. These methods include:  • %LET statement  • macro parameters (named and positional)  • iterative %DO statement  • using the INTO in PROC SQL  • using the CALL SYMPUTX routine  **%LET statement:**  It is roughly the macro language equivalent of the of the DATA step's assignment statement.  The %LET statement is followed by the macro variable name, an equal sign (=), and then the text value to be assigned to the macro variable.  The syntax of the %LET statement is  %LET *macro-variable-name = text-or-text-value*;  **Parameter of a macro :**  When a macro is defined with parameters, the macro variables that are created are always local to the macro. If they are not passed a value during the macro call, and do not have an initial value (as keyword parameters can have), then the macro variable has a null value (length = 0).  %MACRO test(mac =);  <macro statements>  %MEND test;  In this case, if the macro %test is called without giving a value to the macro variable "mac", then "mac" has an initial value of null, with a length of 0.  **Iterative %DO statement:**  When you use a %DO loop in a macro, if the macro variable has not been defined previously, this will create it.  Syntax :  %DO *macro-variable = start %TO stop <%by increament>;*  *… text …*  *%End;*  As an example:  %MACRO test;  %DO cnt = 1 %TO 5;  <macro statements>  %end; %\* do cnt loop;  %MEND test;  In this case, the %DO statement creates a macro variable called "cnt" if it does not already exist.  We may consider doing two things:  1. Always using a %LOCAL statement explicitly declaring the loop counter as a local variable.  2. Using the declared macro variable only as a loop counter, so you can always keep track of it and know what variable(s) is/are being used as loop counters.  **INTO IN PROC SQL :**  PROC SQL INTO Clause The INTO clause of PROC SQL creates macro variables, and the attribute of extra blanks depends on how the variables are created.  For instance, this will create one macro variable:  PROC SQL;  SELECT varx  INTO :macx  FROM work.test;  QUIT;  This will create a single macro variable called "macx", keeping any leading or trailing blanks. If varx is a number, the macro variable will be right-justified and 8 characters long, including leading spaces.  **using the CALL SYMPUTX routine :**  If *macro-variable* does not exist, SYMPUT creates it. SYMPUT makes a macro variable assignment when the program executes.  **Syntax :**  **CALL SYMPUT**(*macro-variable*, *value*);  ***macro-variable***  can be one of the following items:  1.a character string that is a SAS name, enclosed in quotation marks.  2.the name of a character variable whose values are SAS names.  3.a character expression that produces a macro variable name.  ***VALUE***  It is the value to be assigned, which can be   1. a string enclosed in quotation marks. 2. the name of a numeric or character variable. 3. a DATA step expression.   **3.How to call a macro variable ?**  We call the macro variable with % macro\_name(var)  **4.Difference between Global and local macro variables.**  **Global macro variables:**  Global macro variable can be created and used anywhere in the application. The %GLOBAL statement Creates macro variables that are available during the execution of an entire SAS session.   |  | | --- | | Syntax: |  |  | | --- | | **%GLOBAL** *macro-variable-1 <...macro-variable-n>*; |   **Local macro variables:**  Local macro variable can be created and used only inside of the macro block.  %LOCAL Creates macro variables that are available only during the execution of the macro where they are defined.  Syntax:  %LOCAL *macro-variable-1 <...macro-variable-n>*;  **5.How to call other sas programmes ?**  We call other sas programmes through %include statement. When we execute a program that contains the %INCLUDE statement, SAS executes the code, including any statements or data lines that we bring into the program with %INCLUDE.   |  | | --- | | Syntax: |  |  | | --- | | **%INCLUDE** *source(s)*  </<SOURCE2> <S2=*length*> <*operating-environment-options*>>; | |  | | **6.what is the difference between sas functions and user defined functions ?**  **7.Define EVAL,SYSEVALF.**  The %EVAL function evaluates integer arithmetic or logical expressions.  %EVAL operates by converting its argument from a character value to a numeric or logical expression. Then, it performs the evaluation. Finally, %EVAL converts the result back to a character value and returns that value.   |  | | --- | | Syntax: |  |  | | --- | | **%EVAL** (*arithmetic or logical expression*) |   The %SYSEVALF function evaluates arithmetic and logical expressions using floating-point arithmetic and returns a value that is formatted using the BEST32. format. The result of the evaluation is always text. % SYSEVALF is the only macro function that can evaluate logical expressions that contain floating point or missing values.   |  | | --- | | Syntax: |  |  | | --- | | **%SYSEVALF**(*expression*<, *conversion-type*>) | | |