* **Character Functions**

data newlist; set newdata.maillist;

/\* Extract month, day and year \*/

/\* from the date character vara \*/

m = scan(date,1,’ ‘); d = scan(date,2,’ ‘);

y = scan(year,2,’,’);

dd = compress(d||m||y,’ ,’);

/\* Convert mon, day, year into \*/

/\* new date variableb \*/

newdate = input(dd,date9.); run;

Data cleandata; Set dirtydata;

a = substr(oldid,length(oldid)-3);

put a;

run;

data cleandata; set dirtydata;

oldidx = upcase(oldid);

a = substr(oldid,index(oldidx,’B’),3);

put a; run;

**CAT Function**

**Does not remove leading or trailing blanks, and returns a concatenated character string.**

data \_null\_;

x=' The 2002 Olym';

y='pic Arts Festi';

z=' val included works by D ';

a='ale Chihuly.';

result=cat(x,y,z,a);

put result $char.;

run;

|  |  |
| --- | --- |
| [CATS Function](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a002256540.htm) | Removes leading and trailing blanks, and returns a concatenated character string. |
|  | [CATT Function](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a002257057.htm) | Removes trailing blanks, and returns a concatenated character string. |
|  | [CATX Function](http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/a002257076.htm) | Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string. |

|  |  |
| --- | --- |
| **Function** | **Equivalent Code** |
| CAT(OF X1-X4) | X1||X2||X3||X4 |
| CATS(OF X1-X4) | TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))||  TRIM(LEFT(X4)) |
| CATT(OF X1-X4) | TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4) |
| CATX(SP, OF X1-X4) | TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP||  TRIM(LEFT(X3))||SP||TRIM(LEFT(X4)) |

data \_null\_;

separator='%%$%%';

x='The Olympic ';

y=' Arts Festival ';

z=' includes works by ';

a='Dale Chihuly.';

result=catx(separator,x,y,z,a);

put result $char.;

run;

**COMPRESS Function**

**Returns a character string with specified characters removed from the original string.**

**LEFT Function**

**Left-aligns a character string.**

**TRIM Function**

**Removes trailing blanks from a character string, and returns one blank if the string is missing.**

**COMPBL Function**

**Removes multiple blanks from a character string.**

|  |  |
| --- | --- |
| string='125 E Main St';  length address $10;  address=compbl(string);  put address; | 125 E Main |

**FIND Function**

**Searches for a specific substring of characters within a character string.**

The FIND function searches *string* for the first occurrence of the specified *substring*, and returns the position of that substring. If the substring is not found in *string*, FIND returns a value of 0.

|  |
| --- |
| **Comparisons** |

* The FIND function searches for substrings of characters in a character string, whereas the FINDC function searches for individual characters in a character string.
* The FIND function and the INDEX function both search for substrings of characters in a character string. However, the INDEX function does not have the *modifiers* nor the *startpos* arguments.

**INDEX Function**

**Searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string.**

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

|  |  |
| --- | --- |
| **Tip:** | Enclose a literal string of characters in quotation marks. |
| **Tip:** | Both leading and trailing spaces are considered part of the ***excerpt*** argument. To remove trailing spaces, include the TRIM function with the ***excerpt*** variable inside the INDEX function. |

data \_null\_;

length a b $14;

a='ABC.DEF (X=Y)';

b='X=Y';

q=index(a,b);

w=index(a,trim(b));

put q= w=;

run;

SAS writes the following output to the log:

q=0 w=10

# LENGTH Function

**Returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string.**

# LOWCASE Function

**Converts all letters in an argument to lowercase.**

In a DATA step, if the LOWCASE function returns a value to a variable **that has not previously been assigned a length**, then that variable is given the length of the argument.

# UPCASE Function

**Converts all letters in an argument to uppercase.**

# PROPCASE Function

**Converts all words in an argument to proper case.**

The PROPCASE function copies a character argument and converts all uppercase letters to lowercase letters. It then converts to uppercase the first character of a word that is preceded by a blank, forward slash, hyphen, open parenthesis, period, or tab. PROPCASE returns the value that is altered.

data \_null\_;

input place $ 1-40;

name=propcase(place);

put name;

datalines;

INTRODUCTION TO THE SCIENCE OF ASTRONOMY

VIRGIN ISLANDS (U.S.)

SAINT KITTS/NEVIS

WINSTON-SALEM, N.C.

;

run;

SAS writes the following output to the log:

Introduction To The Science Of Astronomy

Virgin Islands (U.S.)

Saint Kitts/Nevis

Winston-Salem, N.C.

# SCAN Function

**Returns the *n*th word from a character string.**

### Example 1: Finding the First and Last Words in a String

The following example scans a string for the first and last words. Note the following:

* A negative count instructs the SCAN function to scan from right to left.
* Leading and trailing delimiters are ignored because the M modifier is not used.
* In the last observation, all characters in the string are delimiters.

options pageno=1 nodate ls=80 ps=64;

data firstlast;

input String $60.;

First\_Word = scan(string, 1);

Last\_Word = scan(string, -1);

datalines4;

Jack and Jill

& Bob & Carol & Ted & Alice &

Leonardo

! $ % & ( ) \* + , - . / ;

;;;;

proc print data=firstlast;

run;

Results of Finding the First and Last Words in a String

The SAS System 1

First\_ Last\_

Obs String Word Word

1 Jack and Jill Jack Jill

2 & Bob & Carol & Ted & ALice & Bob Alice

3 Leonardo Leonardo Leonardo

4 ! $ % & ( ) \* + , - . / ;

### Example 2: Finding All Words in a String

data all;

length word $20;

drop string;

string = ' The quick brown fox jumps over the lazy dog. ';

do until(word=' ');

count+1;

word = scan(string, count);

output;

end;

run;

proc print data=all noobs;

run;

Results of Finding All Words without Using the M Modifier

The SAS System 1

word count

The 1

quick 2

brown 3

fox 4

jumps 5

over 6

the 7

lazy 8

dog 9

10

# STRIP Function

**Returns a character string with all leading and trailing blanks removed.**

|  |
| --- |
| **Comparisons** |

The following list compares the STRIP function with the TRIM function:

* For strings that are blank, the STRIP function return a string with a length of zero, whereas the TRIM function returns a single blank.
* For strings that lack leading blanks but have at least one non-blank character, the STRIP and TRIM functions return the same value.

data lengthn;

input string $char8.;

original = '\*' || string || '\*';

stripped = '\*' || strip(string) || '\*';

datalines;

abcd

abcd

abcd

abcdefgh

x y z

;

proc print data=lengthn;

run;

Results from the STRIP Function

The SAS System 1

Obs string original stripped

1 abcd \*abcd \* \*abcd\*

2 abcd \* abcd \* \*abcd\*

3 abcd \* abcd\* \*abcd\*

4 abcdefgh \*abcdefgh\* \*abcdefgh\*

5 x y z \* x y z \* \*x y z\*

# TRIM Function

**Removes trailing blanks from a character string, and returns one blank if the string is missing.**

If the argument is blank, TRIM returns one blank.

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| x="A"||trim(" ")||"B"; put x; | A B |
| x=" "; y=">"||trim(x)||"<"; put y; | > < |

# TRANSLATE Function

**Replaces specific characters in a character string.**

In a DATA step, if the TRANSLATE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument

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

# TRANWRD Function

**Replaces all occurrences of a substring in a character string.**

The TRANWRD function replaces all occurrences of a given substring within a character string. The TRANWRD function does not remove trailing blanks in the *target* string and the *replacement* string.

|  |
| --- |
| **Comparisons** |

The TRANSLATE function converts every occurrence of a user-supplied character to another character. TRANSLATE can scan for more than one character in a single call. In doing this scan, however, TRANSLATE searches for every occurrence of any of the individual characters within a string. That is, if any letter (or character) in the target string is found in the source string, it is replaced with the corresponding letter (or character) in the replacement string.

The TRANWRD function differs from TRANSLATE in that TRANWRD scans for substrings and replaces those substrings with a second substring.

name=tranwrd(name, "Mrs.", "Ms.");

name=tranwrd(name, "Miss", "Ms.");

put name;

|  |  |
| --- | --- |
| **Values** | **Results** |
| Mrs. Joan Smith | Ms. Joan Smith |
| Miss Alice Cooper | Ms. Alice Cooper |

**Removing Repeated Commas**

data \_null\_;

mytxt='If you exercise your power to vote,,,then your opinion will be heard,,';

newtext=tranwrd(mytxt, ',,,', ',');

newtext2=tranwrd(newtext, ',,' , '.');

put // mytxt= / newtext= / newtext2=;

run;

Output from Removing Repeated Commas

mytxt=If you exercise your power to vote,,,then your opinion will be heard,,

newtext=If you exercise your power to vote,then your opinion will be heard,,

newtext2=If you exercise your power to vote,then your opinion will be heard.

* **Date and Time Functions:**

# DATEPART Function

**Extracts the date from a SAS datetime value.**

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| conn='01feb94:8:45'dt;  servdate=datepart(conn);  put servdate worddate.; | February 1, 1994 |

# DAY Function

**Returns the day of the month from a SAS date value.**

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| now='05may97'd;  d=day(now);  put d; | 5 |

# MONTH Function

**Returns the month from a SAS date value.**

|  |  |
| --- | --- |
| date='25jan94'd;  m=month(date);  put m; | 1 |

# YEAR Function

**Returns the year from a SAS date value.**

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| date='25dec97'd;  y=year(date);  put y; | 1997 |

# INTCK Function

**Returns the count of the number of interval boundaries between two dates, two times, or two datetime values.**

**INTCK**(*interval*, *start-from*, *increment*)

Types of intervals: <http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a003065889.htm#a003065892>

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| qtr=intck('qtr','10jan95'd,'01jul95'd);  put qtr; | 2 |
| year=intck('year','31dec94'd,  '01jan95'd);  put year; | 1 |
| year=intck('year','01jan94'd,  '31dec94'd);  put year; | 0 |
| semi=intck('semiyear','01jan95'd,  '01jan98'd);  put semi; | 6 |
| weekvar=intck('week2.2','01jan97'd,  '31mar97'd);  put weekvar; | 7 |
| wdvar=intck('weekday7w','01jan97'd,  '01feb97'd);  put wdvar; | 26 |
| y='year';  date1='1sep1991'd;  date2='1sep2001'd;  newyears=intck(y,date1,date2);  put newyears; | 10 |
| y=trim('year ');  date1='1sep1991'd + 300;  date2='1sep2001'd - 300;  newyears=intck(y,date1,date2);  put newyears; | 8 |

# DATDIF Function

**Returns the number of days between two dates after computing the difference between the dates according to specified day count conventions.**

# YRDIF Function

**Returns the difference in years between two dates.**

# INTNX Function

**Increments a date, time, or datetime value by a given time interval, and returns a date, time, or datetime value.**

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

**'*alignment*'**

controls the position of SAS dates within the interval. You must enclose ***alignment*** in quotation marks. ***Alignment*** can be one of these values:

BEGINNING B

specifies that the returned date or datetime value is aligned to the beginning of the interval.

MIDDLE M

specifies that the returned date or datetime value is aligned to the midpoint of the interval, which is the average of the beginning and ending alignment values.

END E

specifies that the returned date or datetime value is aligned to the end of the interval.

SAME

specifies that the date that is returned has the same alignment as the input date.

Types of intervals: <http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a003065889.htm#a003065892>

x=intnx('week', '17oct03'd, 6);

put x date9.;

INTNX returns the value 23NOV2003.

intnx('week', '15mar2000'd, 1, 'same'); returns 22MAR2000

intnx('dtweek', '15mar2000:8:45'dt, 1, 'same'); returns 22MAR00:08:45:00

intnx('year', '15mar2000'd, 5, 'same'); returns 15MAR2005

|  |  |
| --- | --- |
| **SAS Statements** | **Results** |
| date1=intnx('month','01jan95'd,5,'beginning');  put date1 / date1 date7.; | 12935  01JUN95 |
| date2=intnx('month','01jan95'd,5,'middle');  put date2 / date2 date7.; | 12949  15JUN95 |
| date3=intnx('month','01jan95'd,5,'end');  put date3 / date3 date7.; | 12964  30JUN95 |
| date4=intnx('month','01jan95'd,5,'sameday');  put date4 / date4 date7.; | 12935  01JUN95 |
| date5=intnx('month','15mar2000'd,5,'same');  put date5 / date5 date9.; | 14837  15AUG2000 |
| interval='month';  date='1sep2001'd;  align='m';  date4=intnx(interval,date,2,align);  put date4 / date4 date7.; | 15294  15NOV01 |
| x1='month ';  x2=trim(x1);  date='1sep2001'd + 90;  date5=intnx(x2,date,2,'m');  put date5 / date5 date7.; | 15356  16JAN02 |

**Conversion Functions:**

**PUT Function**

**Returns a value using a specified format.**

Use PUT to convert a numeric value to a character value. You cannot use the PUT function to change the type of a variable in a data set from numeric to character.

### Using PUT and INPUT Functions

numdate=122591;

chardate=put(numdate,z6.);

sasdate=input(chardate,mmddyy6.);

# INPUT Function

The INPUT function enables you to convert the value of *source* by using a specified informat. The informat determines whether the result is numeric or character. Use INPUT to convert character values to numeric values or other character values.

**LAG Function**

**Returns values from a queue.**

data one;

input x @@;

y=lag1(x);

z=lag2(x);

datalines;

1 2 3 4 5 6

;

proc print data=one;

title 'LAG Output';

run;

Output from Generating Two Lagged Values

LAG Output 1

Obs x y z

1 1 . .

2 2 1 .

3 3 2 1

4 4 3 2

5 5 4 3

6 6 5 4

### Generating Multiple Lagged Values in BY-Groups

data old;

input start end;

datalines;

1 1

1 2

1 3

1 4

1 5

1 6

1 7

2 1

2 2

3 1

3 2

3 3

3 4

3 5

;

data new(drop=i count);

set old;

by start;

/\* Create and assign values to three new variables. Use ENDLAG1- \*/

/\* ENDLAG3 to store lagged values of END, from the most recent to the \*/

/\* third preceding value. \*/

array x(\*) endlag1-endlag3;

endlag1=lag1(end);

endlag2=lag2(end);

endlag3=lag3(end);

/\* Reset COUNT at the start of each new BY-Group \*/

if first.start then count=1;

/\* On each iteration, set to missing array elements \*/

/\* that have not yet received a lagged value for the \*/

/\* current BY-Group. Increase count by 1. \*/

do i=count to dim(x);

x(i)=.;

end;

count + 1;

run;

proc print;

run;

Output from Generating Three Lagged Values

The SAS System 1

Obs start end endlag1 endlag2 endlag3

1 1 1 . . .

2 1 2 1 . .

3 1 3 2 1 .

4 1 4 3 2 1

5 1 5 4 3 2

6 1 6 5 4 3

7 1 7 6 5 4

8 2 1 . . .

9 2 2 1 . .

10 3 1 . . .

11 3 2 1 . .

12 3 3 2 1 .

13 3 4 3 2 1

14 3 5 4

**Call Routines:**

*CALL routine* alters variable values or performs other system functions. CALL

routines are similar to functions, but differ from functions in that you cannot use them

in assignment statements or expressions.

**DO LOOPS :**

There are four forms of the DO statement:

1. The DO statement designates a group of statements that are to be executed as a unit, usually as a part of IF-THEN/ELSE statements.

if years>5 then

do;

months=years\*12;

put years= months=;

end;

1. The iterative DO statement executes a group of statements repetitively based on the value of an index variable. If you specify an UNTIL clause or a WHILE clause, then the execution of the statements is also based on the condition that you specify in the clause.

The iterative DO loop executes the statements between DO and END repetitively based on the value of an index variable.

DO *index-variable* = *start* TO *stop* <BY *increment*>;

dcl num k=18 n=11;

do i=k+2 to n-1 by -2;

put i;

end;

The following example uses an UNTIL clause to set a flag, and then it checks the flag during each iteration of the loop:

flag=0;

do i=1 to 10 until(flag);

*...SAS statements...*

if *expression* then flag=1;

end;

The following loop executes as long as I is within the range of 10 to 0 and MONTH is equal to **JAN**.

do i=10 to 0 by -1 while(month='JAN');

*...SAS statements...*

end;

1. The DO UNTIL statement executes a group of statements repetitively until the condition that you specify is true. The condition is checked **after** each iteration of the loop.

n=0;

do while(n<5);

put n=;

n+1;

end;

1. The DO WHILE statement executes a group of statements repetitively as long as the condition that you specify remains true. The condition is checked **before** each iteration of the loop.

n=0;

do until(n>=5);

put n=;

n+1;

end;

### Example 1: Using Various Forms of the Iterative DO Statement

* These iterative DO statements use a list of items for the value of *start*:
  + do month='JAN','FEB','MAR';
  + do count=2,3,5,7,11,13,17;
  + do i=5;
  + do i=var1, var2, var3;
  + do i='01JAN2001'd,'25FEB2001'd,'18APR2001'd;
* These iterative DO statements use the *start* TO *stop* syntax:
  + do i=1 to 10;
  + do i=1 to exit;
  + do i=1 to x-5;
  + do i=1 to k-1, k+1 to n;
  + do i=k+1 to n-1;
* These iterative DO statements use the BY *increment* syntax:
  + do i=n to 1 by -1;
  + do i=.1 to .9 by .1, 1 to 10 by 1,
  + 20 to 100 by 10;
  + do count=2 to 8 by 2;
* These iterative DO statements use WHILE and UNTIL clauses:
  + do i=1 to 10 while(x<y);
  + do i=2 to 20 by 2 until((x/3)>y);
  + do i=10 to 0 by -1 while(month='JAN');
* In this example, the DO loop is executed when I=1 and I=2; the WHILE condition is evaluated when I=3, and the DO loop is executed if the WHILE condition is true.

DO I=1,2,3 WHILE (condition);

**Controlling DO Loops:**

The **LEAVE** statement stops processing the current DO loop and resumes with the next statement after the DO loop. With the LEAVE statement, you have the option of specifying a label for the DO statement:

do i=1 to n by m;

...*more SAS statements*...

if i=10 then leave;

end;

if i=10 then put 'EXITED LOOP';

If you have nested DO loops and you want to skip out of more than one loop, you can specify the label of the loop that you want to leave. For example, the following LEAVE statement causes execution to skip to the last PUT statement:

myloop:

do i=1 to 10;

do j=1 to 10;

if j=5 then leave myloop;

put i= j=;

end;

end;

put 'this statement executes next';

return;

**Validating and Cleaning data**

**http://www.biostat.umn.edu/~greg-g/PH5420/m237\_14\_a.pdf**