PharmaSUG 2016 - Paper TT10

May the Function Be With You: Helpful SAS® Functions, Particularly When Handling CDISC Data

Angela Lamb, Chiltern, King of Prussia, PA

ABSTRACT

SAS® Functions are a basic component of the DATA step, but some lesser-known functions are often overlooked, leading to lengthy, complicated, and unnecessary code. Programmers can get stuck in a rut, habitually handling data one way, while a more efficient and more robust function could simplify the process. CDISC data presents us with unique programming challenges. One such challenge is the need to handle long, ISO 8601 formatted dates. Functions can be very helpful in processing these dates—for instance, in breaking them apart to calculate durations. This paper explores the basics of SAS functions, some that are especially useful for handling CDISC data, and functions that are lesser-known but invaluable to the clinical trials programmer.

INTRODUCTION

A SAS function performs a computation or system manipulation on arguments, and returns a value that can be used in an assignment statement or elsewhere in expressions. It is a built-in piece of code that changes a variable or argument in some way. Functions can be used to locate text within a character variable, calculate a sum, concatenate variables, and for many other situations. Often, functions simplify a program by replacing long strings of code with one short line. We will examine several functions that are useful to have on your tool belt as a programmer.

Call routines are very similar to functions, but cannot be used in assignment statements. We will also look at one useful call routine in handling ISO 8601 dates.

HOW ARE FUNCTIONS USED IN SAS CODE?

Functions are often used in the DATA step. Most functions require one or more arguments.

The basic syntax of a function is as follows:

```
function-name(argument1,argument2)
```

However, some functions do not require any arguments. One example is the TODAY function. This function returns today's date without needing any arguments included in the parentheses.

```
x=today();
```

USEFUL FUNCTIONS

While there are over 190 functions available in the SAS system, we will cover some useful, but often overlooked functions that aid in everyday programming.

MISSING

Code similar to the following is frequently used by programmers.

```
if charvar='' then …
if numvar= . then …
```

The MISSING function can be used in both of these situations, eliminating the need to know whether the variable is numeric or character.

```
if missing(charvar) then …
if missing(numvar) then …
```

This not only can save time while programming, but the MISSING function makes your program more dynamic, as it will work whether a numeric or character variable is passed through it.

SCAN

SCAN is a very useful function. It gives you the *nth* word from a character string, and various delimiters can be specified to separate the "words", or pieces of text, searched for.

Example 1:

```
data yoda;
  text='Try not. Do, or do not. There is no try.';
  text2=scan(text, 2);
  put text2=;
run;
```

```
TEXT2=not
```

Output 1. Output from Example 1 DATA step.

Here, the SCAN function selects the second word in the variable TEXT, as delimited by any of the default delimiters (blank ! $ % & ( ) * + < - . / ; < ^ |).

However, you may also specify which delimiters you would like SCAN to recognize.

Example 2:

```
data yoda2;
  text='Try not. Do, or do not. There is no try.';
  text2=scan(text, 2, '.');

  put text2=;
run;
```

```
TEXT2= Do, or do not
```

Output 2. Output from Example 2 DATA step.

As shown in example 2, SCAN can be used with only certain delimiters, so it can be used to select an entire phrase or sentence. Here, the second phrase was specified, as delimited by a period (.).

Modifiers can also be added as options within the SCAN function. Some modifiers include:

- b = scan backward from right to left instead of left to right
- r = remove leading and trailing blanks from the word or phrase that SCAN returns
- t = trim trailing blanks

Example 3:

```
data yoda3;
  text='Try not. Do, or do not. There is no try.';
  text2=scan(text, 2, '.', 'r');
  text3=scan(text, 1, '.', 'br');

  put text2=;
  put text3=;
run;
```

```
TEXT2=Do, or do not
TEXT3=There is no try
```

Output 3. Output from Example 3 DATA step.

As you can see, when we ran the code from Example 2, the output contained a leading blank. Using the modifier 'r' removed the leading blank from the phrase "Do, or do not".

More than one modifier can be used at a time within the SCAN function by stringing them together. For the variable TEXT3, we used both 'b' and 'r'. The modifier 'b' told SCAN to scan backwards, while 'r' told SCAN to drop the leading and trailing blanks. We are left with "There is no try" since it is the first phrase scanned from right to left using only the '.' delimiter.

When working with CDISC data, VISITNUM may simply be the numeric part of VISIT, but the numeric portion may not always be in the same location within VISIT. Instead of a string of if-then-else statements, the SCAN function could be used to locate the visit number. Given visits such as “VISIT 1” or “UNSCHEDULED VISIT 2.01”, where the number is the last part of the visit, SCAN could be used to assign the visit number.

```
visitnum=input(scan(visit, 1, ' ', 'b'),best.);
```

CATX

The CATX function can replace lengthy concatenation code. CATX concatenates the variables or strings specified, automatically stripping the items and converting them to character if they are numeric using the BESTw. format.

The syntax for the CATX function is CATX(*delimiter, item-1* <,…*item-n*>).

Example 4

```
data jedi;
  yoda=' Yoda';
  luke='Luke Skywalker';
  obi='Obi-Wan Kenobi ';

  **old way**;
  jedi_list1=strip(yoda)||', '||strip(luke)||', '||strip(obi);

  **new way using catx**;
  jedi_list2=catx(', ',yoda,luke,obi);

  put jedi_list1=;
  put jedi_list2=;
run;
```

```
JEDI_LIST1=Yoda, Luke Skywalker, Obi-Wan Kenobi
JEDI_LIST2=Yoda, Luke Skywalker, Obi-Wan Kenobi
```

Output 4. Output from Example 4 DATA step.

You can see that both the traditional concatenation method and the new method using CATX produce the same result, but the CATX code is much simpler.

TRANSTRN

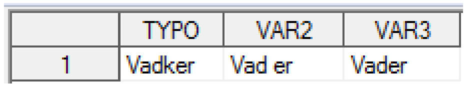
TRANSTRN is a simple function that can be used in a variety of situations. It replaces a string of characters with another string. The replacement string does not have to be the same length as the source string. This is particularly helpful when you have a long string of text with the source string randomly placed in unpredictable locations throughout.

Or—unlike its cousin TRANWRD—TRANSTRN can replace a string with nothing (TRANWRD will leave a blank space behind).

The syntax for both TRANWRD and TRANSTRN is FUNCTION(*variable, ‘string to find’, ‘string to replace it with’*). If you want to replace the string with nothing and just remove the string to find, then use TRANSTRN and use STRIP(‘’) as your replacement string.

Example 5

```
data darth;
   typo='Vadker';
   var2=tranwrd(typo,'k','');
   var3=transtrn(typo,'k',strip(''));
run;
```



	TYPO	VAR2	VAR3
1	Vadker	Vad er	Vader

Display 1. Dataset created from Example 5 DATA step

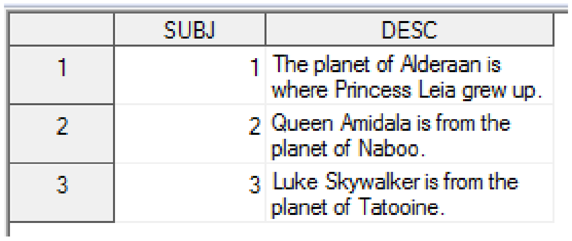
As you can see, the TRANSTRN function has the ability to leave no blank when used to remove a string, when combined with the STRIP function. When the TRANWRD function was used, a space was left behind where the letter "k" initially was, as shown in VAR2. However, TRANSTRN did not leave a space behind, but rather replaced the "k" with nothing, since we used the STRIP function around the replacement string.

Searching for and removing text using these functions can be very handy in multiple situations; for example, if you need to search for a drug name within an inclusion/exclusion criterion text and replace it with the generic name.

Example 6

Let's look at another example using TRANSTRN.

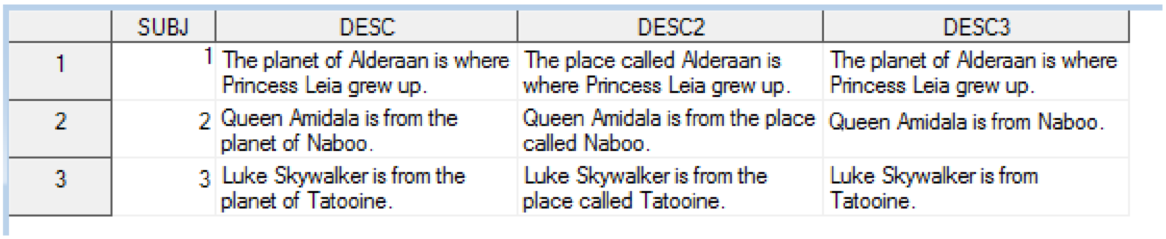
```
data planets;
  subj=1;
  desc='The planet of Alderaan is where Princess Leia grew up.';
  output;

  subj=2;
  desc='Queen Amidala is from the planet of Naboo.';
  output;

  subj=3;
  desc='Luke Skywalker is from the planet of Tatooine.';
  output;
run;
```



	SUBJ	DESC
1	1	The planet of Alderaan is where Princess Leia grew up.
2	2	Queen Amidala is from the planet of Naboo.
3	3	Luke Skywalker is from the planet of Tatooine.

Display 2. Dataset created from Example 6 PLANETS DATA step

```
data planets2;
  set planets;
  desc2=transtrn(desc, 'planet of', 'place called');
  desc3=transtrn(desc, 'the planet of ', strip(''));
run;
```



	SUBJ	DESC	DESC2	DESC3
1	1	The planet of Alderaan is where Princess Leia grew up.	The place called Alderaan is where Princess Leia grew up.	The planet of Alderaan is where Princess Leia grew up.
2	2	Queen Amidala is from the planet of Naboo.	Queen Amidala is from the place called Naboo.	Queen Amidala is from Naboo.
3	3	Luke Skywalker is from the planet of Tatooine.	Luke Skywalker is from the place called Tatooine.	Luke Skywalker is from Tatooine.

Display 3. Dataset created from Example 6 PLANETS2 DATA step

More than one word can be replaced, as the variable DESC2 shows. We replaced the text "planet of" with "place called", and the replacement was done no matter where the string was located within the text.

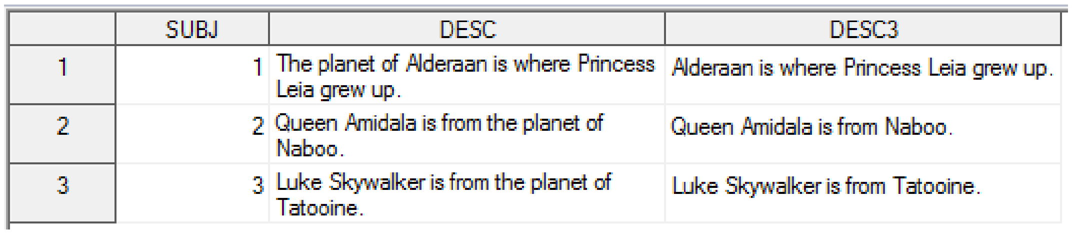
However, it is important to keep in mind that it is case-sensitive. For the variable DESC3, we substituted "the planet of " with nothing, so that the phrase was removed. You can see that the substitution did not work for the first observation because the case did not match on the word "The".

One solution to this, if you are working with data where the case is potentially different in different observations, is to run the function multiple times on the same variable, to catch the variations.

```
data planets3;
  set planets;
  desc3=transtrn(desc, 'the planet of ', strip(''));
```

```
    desc3=transtrn(desc3, 'The planet of ', strip(''));
  run;
```

This will catch the different variations of case used within the variable DESC. In the dataset below, all the desired replacements were made.



	SUBJ	DESC	DESC3
1	1	The planet of Alderaan is where Princess Leia grew up.	Alderaan is where Princess Leia grew up.
2	2	Queen Amidala is from the planet of Naboo.	Queen Amidala is from Naboo.
3	3	Luke Skywalker is from the planet of Tatooine.	Luke Skywalker is from Tatooine.

Display 4. Dataset created from Example 6 PLANETS3 DATA step

ANYDIGIT/ANYALPHA

A couple of more recent additions to the function catalog of SAS are the functions ANYDIGIT and ANYALPHA. ANYDIGIT searches a string for the first occurrence of any character that is a digit, and ANYALPHA searches for the first occurrence of any upper or lowercase letter. If such a character is found, the function gives the position in the string of that character. If no such character is found, the function returns a value of 0.

The syntax of these two functions is as follows:

ANYDIGIT(*string*< ,*start*>)

ANYALPHA(*string* <,*start*>)

where *string* is the character constant, variable, or expression to search.

The second argument is optional, but if supplied, it can tell the function where to begin the search and which direction to search. A negative number can be supplied as the <*start*> value to indicate that the search should go backwards. If a negative <*start*> value is supplied that is beyond the length of the string, the search will simply begin at the end of the string. These functions are handy because they not only tell you *if* a string contains a digit or a letter, but *where* it occurs.

Example 7

```
  data han;
    bio='Ford was born in 1942.';
    bio2='Harrison Ford was born in Chicago, IL.';

    test1=anydigit(bio);
    test2=anydigit(bio,-100);
    test3=anydigit(bio2);

    put test1= test2= test3=;
 run;
```

```
TEST1=18 TEST2=21 TEST3=0
```

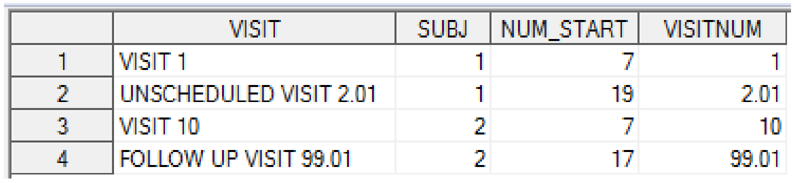
Output 5. Output from Example 7 DATA step.

You can see that TEST1 contains the location of the first digit in the string. TEST2 runs the search backwards, and begins at the end of the string, since 100 is greater than the number of characters in the string. The first digit reading from right to left is character number 21. TEST3 returns 0, because no digit is found in the second variable, BIO2.

Example 8

Revisiting the creation of VISITNUM, we can use the ANYDIGIT function in the creation of the variable VISITNUM from the variable VISIT. Instead of using SCAN, we could use ANYDIGIT to find where the numeric portion of the value started and then use that to split out the visit number.

```
  data anydigit;
    length visit $100;
    subj=1; visit='VISIT 1'; output;
    subj=1; visit='UNSCHEDULED VISIT 2.01'; output;
    subj=2; visit='VISIT 10'; output;
    subj=2; visit='FOLLOW UP VISIT 99.01'; output;
  run;

  data anydigit2;
    set anydigit;

    num_start=anydigit(visit);
    visitnum=input(substr(visit,num_start),best.);
  run;
```



	VISIT	SUBJ	NUM_START	VISITNUM
1	VISIT 1	1	7	1
2	UNSCHEDULED VISIT 2.01	1	19	2.01
3	VISIT 10	2	7	10
4	FOLLOW UP VISIT 99.01	2	17	99.01

Display 5. Dataset created from Example 8 ANYDIGIT2 DATA step

The new variable VISITNUM contains only the numeric portion of the variable VISIT, obtained in part by using the ANYDIGIT function to locate the first digit in the variable.

DATE FUNCTIONS

Date functions can greatly simplify programming code when working with date variables. CDISC, primarily SDTM, requires using dates in the ISO 8601 format, which we often have to break apart to calculate duration in weeks, months, and years, as well as use just the date and time portion. Instead of having to put a date into a character format, substring a particular section, and convert back to numeric, utilizing date functions makes this process quick and easy. Date functions and call routines are also very helpful while performing QC checks.

DAY/MONTH/YEAR

To break apart a numeric date variable and only capture the day portion, the DAY function can be used. This can replace having to put, substring, and input the variable to obtain the day.

Old method:
```
  dayvar=input(substr(put(startdt,yymmdd10.),9,2),best.);
```

Or more simply:
```
  dayvar=day(startdt);
```

The MONTH and YEAR functions work similarly to return a numeric variable containing the month or year, respectively.

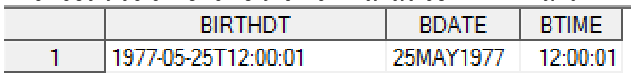
DATEPART/TIMEPART

The functions DATEPART and TIMEPART are useful when dealing with ISO 8601 dates. To obtain just the date portion or just the time portion, try using these functions.

Example 9

```
  data starwars;
    **start by creating a numeric datetime*;
    birthdt='25MAY1977T12:00:01'dt;

    **now we will select just the date part or time part*;
    bdate=datepart(birthdt);
    btime=timepart(birthdt);

    format birthdt e8601dt19. bdate date9. btime time8.;
  run;
```

The result below shows the new variables BDATE and BTIME.



	BIRTHDT	BDATE	BTIME
1	1977-05-25T12:00:01	25MAY1977	12:00:01

Display 6. Dataset created from Example 9 DATA step

MDY

Instead of breaking a date apart, the MDY function will **combine** month, day, and year values to make a SAS date. You may supply either constants or variables as the arguments.

Example 10

```
data episode7;
  mon=12;
  dy=18;
  yr=2015;

  release=mdy(mon,dy,yr);
  format release yymmdd10.;

  put release= ;
run;
```

```
RELEASE=2015-12-18
```

Output 6. Output from Example 10 DATA step.

The date variable RELEASE was made by combining the numeric variables MON, DY, and YR into one numeric date using the MDY function.

DHMS

The DHMS function works like MDY, but the arguments are date, hour, minute, and second, combined to form a single numeric datetime variable.

Example 11

```
data episode7b;
  set episode7;

  release2=dhms(release,8,10,15);
  format release2 e8601dt19.;

  put release2= ;
run;
```

```
RELEASE2=2015-12-18T08:10:15
```

Output 7. Output from Example 11 DATA step.

Here, we combined the former RELEASE numeric date variable with a time portion, using the function DHMS. In Example 10, we used variables as the arguments to the function, but here we used constants, which work as well.

CALL IS8601_CONVERT ROUTINE

The CALL IS8601_CONVERT routine converts an ISO 8601 interval to datetime and duration values, or converts datetime and duration values to an ISO8601 interval. It can be useful when working with CDISC data which often contains these dates, and it can also be useful for performing QC checks on durations.

Here is the syntax of this call routine:
CALL IS8601_CONVERT(*convert-from, convert-to, <from-variables>, <to-variables>, <date_time_replacements>);*

The *convert-from* and *convert-to* arguments can contain

'intvl' (specifies that the value is an interval value)
'dt/du' (specifies that the values are datetime/duration)

‘du/dt’ (specifies that the values are duration/datetime)
‘dt/dt’ (specifies that the values are both datetime)
‘du’ (specifies that the value is a duration)

The *convert-to* argument can also contain

‘start’ (specifies to create a value that is the beginning datetime or duration of an interval)
‘end’ (specifies to create a value that is the ending datetime or duration of an interval)

Example 12

```
data diff;
  episode4='25MAY1977T12:00:01'dt;
  episode7='18DEC2015T8:10:15'dt;

  length dur $16;
  call is8601_convert('dt/dt','du',episode4,episode7,dur);

  put dur= $n8601e.;
run;
```

```
DUR=P38Y6M23DT20H10M14S
```

Output 8. Output from Example 12 DATA step.

As you can see, there 38 years, 6 months, 23 days, 20 hours, 10 minutes, and 14 seconds between EPISODE4 and EPISODE7 in this example.

Example 13

Let’s say you want to take a datetime variable (we will use the datetime EPISODE7, the date Episode 7 of *Star Wars* was released in theaters), and add 12 weeks to it, to get the end date of the movie being shown in a particular theater. The CALL IS8601_CONVERT routine can also be used in this situation. This time our arguments will be a datetime and a duration (dt/du), to be converted to the ending datetime of the interval (end).

```
data dur;
  episode7='18DEC2015T8:10:15'dt;
  inthr='P12w';

  call is8601_convert('dt/du','end',episode7,inthr,newvar);

  put newvar= e8601dt19.;
run;
```

```
NEWVAR=2016-03-11T08:10:15
```

Output 9. Output from Example 13 DATA step.

In this example, we used the call routine to add 12 weeks to our start date and give us the ‘end’ datetime NEWVAR.

This CALL routine is also helpful because if a value is missing for month, day, year, minute, or second, a default date/time replacement will be supplied (1, 1, 0, 0, and 0 respectively). You can also update these replacements to be set to different values by supplying those arguments in the code.

We have only looked at an overview and a few uses of the CALL IS8601_CONVERT routine, and there are many more to be explored. There are helpful guides to be found online to study this CALL routine further. It can be a powerful tool when handling CDISC dates.

CONCLUSION

There are over 190 functions available in BASE SAS for use by the programmer. Functions, and their cousin the CALL routine, are helpful in simplifying programming code, and are powerful tools for the programmer when handling clinical trials data. These pieces of code are not lengthy, and can eliminate the need to “program around” something. For that reason, it can be a good idea to make sure we, as programmers, stay fresh on functions and call routines. It is also worth noting that new functions are often added with subsequent updates to SAS to meet programmers’ need.

REFERENCES

- SAS Institute, Inc. "Functions and CALL Routines." 3/17/2016 Available at http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000245852.htm.

- Wilson, Kim. Copyright 2012. "Harnessing the Power of SAS ISO 8601 Informats, Formats, and the CALL IS8601_CONVERT Routine." *Proceedings of PharmaSUG 2012*. Available at http://www.pharmasug.org/proceedings/2012/DS/PharmaSUG-2012-DS22-SAS.pdf.

- Howard, Neil. "Introduction to SAS Functions." *Proceedings of the SAS Users Group International Conference*. Charlottesville, VA. Available at http://www2.sas.com/proceedings/sugi24/Begtutor/p57-24.pdf.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Angela Lamb
Enterprise: Chiltern International Inc.
Address: 2528 Independence Blvd., Suite 101
City, State ZIP: Wilmington, NC 28412
E-mail: Angela.Lamb@chiltern.com