The Basics of SAS Functions

Definition

SAS *functions* are pre-written routines that perform computations or system manipulations on arguments and return a value. Functions can return either numeric or character results. The value that is returned can be used in an assignment statement or elsewhere in expressions.

Uses of SAS Functions

You can use SAS functions in DATA step programming statements, in WHERE expressions, in macro language statements, in the REPORT procedure, and in Structured Query Language (SQL). They enable you to do the following:

- calculate sample statistics
- create SAS date values
- convert U.S. ZIP codes to state postal codes
- round values
- generate random numbers
- extract a portion of a character value
- convert data from one data type to another

SAS Functions Categories

SAS functions provide programming shortcuts. The following table shows you all of the SAS function categories. This book covers selected functions that convert data, manipulate SAS date values, and modify values of character variables.

Table 14.1 *SAS Function Categories*

Functions by Category			
Arithmetic	Descriptive Statistics*	Numeric*	State and ZIP code*

Functions by Category			
Array	Distance	Probability	Trigonometric
Bitwise Logical Operations	External Files	Quantile	Truncation*
CAS	External Routines	Random Number	Variable Control
Character*	Financial	SAS File I/O	Variable Information
Character String Matching	Hyperbolic	Search	Web Services
Combinatorial	Macro	Sort	Web Tools
Date and Time	Mathematical*	Special*	

* Denotes the functions that are covered in this chapter.

SAS Functions Syntax

Arguments and Variable Lists

To use a SAS function, specify the function name followed by the function arguments, which are enclosed in parentheses.

Syntax, SAS function:

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

Each of the following are *arguments*:

- variables: mean(*x*,*y*,*z*)
- constants: mean(*456*,*502*,*612*,*498*)
- expressions: mean(37*2,192/5,*mean*(*22*,*34*,*56*))

Note: Even if the function does not require arguments, the function name must still be followed by parentheses (for example, *function-name()*).

When a function contains more than one argument, the arguments are usually separated by commas.

```
function-name(argument-1,argument-2,argument-n);
```

Example: Multiple Arguments

Here is an example of a function that contains multiple arguments. Notice that the arguments are separated by commas.

```
mean(x1,x2,x3)
```

The arguments for this function can also be written as a variable list.

```
mean(of x1-x3)
```

Target Variables

A target variable is the variable to which the result of a function is assigned. For example, in the statement below, the variable AvgScore is the target variable.

```
AvgScore=mean(exam1,exam2,exam3);
```

Unless the length of the target variable has been previously defined, a default length is assigned. The default length depends on the function; the default for character functions can be as long as 200.

TIP Default lengths can cause character variables to use more space than necessary in your data set. So, when using SAS functions, consider the appropriate length for any character target variables. If necessary, add a LENGTH statement to specify a length for the character target variable before the statement that creates the values of that variable.

Converting Data with Functions

A Word about Converting Data

The following code automatically converts the variable PayRate from character to numeric.

```
data work.newtemp;
  set cert.temp;
  Salary=payrate*hours;
run;
```

You can also use the INPUT function before performing a calculation. The INPUT function converts character data values to numeric values.

You can use the PUT function to convert numeric data values to character values.

Potential Problems of Omitting INPUT or PUT

If you skip INPUT or PUT function when converting data, SAS detects the mismatched variables and tries an automatic character-to-numeric or numeric-to-character conversion. However, this action is not always successful. Suppose each value of PayRate begins with a dollar sign ($). When SAS tries to automatically convert the values of PayRate to numeric values, the dollar sign blocks the process. The values cannot be converted to numeric values. Similar problems can occur with automatic numeric-to-character conversion.

Therefore, it is a recommended best practice to include INPUT and PUT functions in your programs to avoid data type mismatches and automatic conversion.

Automatic Character-to-Numeric Conversion

By default, if you reference a character variable in a numeric context such as an arithmetic operation, SAS tries to convert the variable values to numeric. For example, in the DATA step below, the character variable PayRate appears in a numeric context. It is multiplied by the numeric variable Hours to create a new variable named Salary.

```
data work.newtemp;
  set cert.temp;
  Salary=payrate*hours;
run;
```

When this step executes, SAS automatically attempts to convert the character values of PayRate to numeric values so that the calculation can occur. This conversion is completed by creating a temporary numeric value for each character value of PayRate. This temporary value is used in the calculation. The character values of PayRate are not replaced by numeric values.

Whenever data is automatically converted, a message is written to the SAS log stating that the conversion has occurred.

Log 14.1 *SAS Log*

```
9246  data work.temp;
9247  set cert.temp;
9248  salary=payrate*hours;
9249  run;

NOTE: Character values have been converted to numeric values at the places given
by:
      (Line):(Column).
      9248:8
NOTE: There were 10 observations read from the data set CERT.TEMP.
NOTE: The data set WORK.TEMP has 10 observations and 16 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

When Automatic Conversion Occurs

Automatic character-to-numeric conversion occurs in the following circumstances:

- A character value is assigned to a previously defined numeric variable, such as the numeric variable Rate.

  ```
  Rate=payrate;
  ```

- A character value is used in an arithmetic operation.

  ```
  Salary=payrate*hours;
  ```

- A character value is compared to a numeric value, using a comparison operator.

  ```
  if payrate>=rate;
  ```

- A character value is specified in a function that requires numeric arguments.

  ```
  NewRate=sum(payrate,raise);
  ```

The following statements are true about automatic conversion.

- It uses the *w.* informat, where *w* is the width of the character value that is being converted.
- It produces a numeric missing value from any character value that does not conform to standard numeric notation (digits with an optional decimal point, leading sign, or scientific notation).

Table 14.2 Automatic Conversion of Character Variables

Character Value	Automatic Conversion	Numeric Value
`12.47`	→	12.47
`-8.96`	→	-8.96
`1.243E1`	→	12.43
`1,742.64`	→	.

Restriction for WHERE Expressions

The WHERE statement does not perform automatic conversions in comparisons. The simple program below demonstrates what happens when a WHERE expression encounters the wrong data type. The variable Number contains a numeric value, and the variable Character contains a character value, but the two WHERE statements specify the wrong data type.

```
data work.convtest;
  Number=4;
  Character='4';
run;
proc print data=work.convtest;
  where character=4;
run;
proc print data=work.convtest;
  where number='4';
run;
```

This mismatch of character and numeric variables and values prevents the program from processing the WHERE statements. Automatic conversion is not performed. Instead, the program stops, and error messages are written to the SAS log.

Log 14.2 *SAS Log*

```
9254 data work.convtest;
9255  Number=4;
9256  Character='4';
9257 run;

NOTE: The data set WORK.CONVTEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

9258 proc print data=work.convtest;
9259  where character=4;
ERROR: WHERE clause operator requires compatible variables.
9260 run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

9261 proc print data=work.convtest;
9262  where number='4';
ERROR: WHERE clause operator requires compatible variables.
9263 run;

NOTE: The SAS System stopped processing this step because of errors.
```

Explicit Character-to-Numeric Conversion

Using the INPUT Function

Use the INPUT function to convert character data values to numeric values. You can explicitly convert the character values of PayRate to numeric values by using the INPUT function.

Syntax, INPUT function:

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

- *source* indicates the character variable, constant, or expression to be converted to a numeric value.
- a numeric *informat* must also be specified, as in this example:

```
input(payrate,2.)
```

When choosing the informat, be sure to select a numeric informat that can read the form of the values.

Table 14.3 *Character Values and Associated Informats*

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

Example: INPUT Function

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. Here is an example of the INPUT function:

```
Test=input(saletest,comma9.);
```

You can use the INPUT function to convert the character values of PayRate to numeric values.

Because PayRate has a length of 2, the numeric informat 2. is used to read the values of the variable.

```
input(payrate,2.)
```

In the following program, the function is added to the assignment statement in the DATA step.

```
data work.newtemp;
  set cert.temp;
  Salary=input(payrate,2.)*hours;
run;
```

After the DATA step is executed, the new data set, which contains the variable Salary, is created. Notice that no conversion messages appear in the SAS log when the INPUT function is used.

Log 14.3 *SAS Log*

```
9272 data work.newtemp;
9273  set cert.temp;
9274  Salary=input(payrate,2.)*hours;
9275 run;

NOTE: There were 10 observations read from the data set CERT.TEMP.
```

Output 14.1 *PROC PRINT Output of Work.NewTemp (partial output)*

Obs	Address		Startdate	Enddate	Payrate	Days	Hours		Dept	Site	Salary
1	65 ELM DR	. . .	19NOV2017	12JAN2018	10	11	88	. . .	DP	26	880
2	11 SUN DR	more	16OCT2017	19NOV2017	8	25	200	more	PURH	57	1600
3	712 HARDWICK STREET	variables	22NOV2017	30DEC2017	40	26	208	variables	PERS	34	8320
4	5372 WHITEBUD ROAD	. . .	29SEP2017	10OCT2017	15	10	80	. . .	BK	57	1200
5	11 TALYN COURT		02NOV2017	13NOV2017	12	9	72		DP	95	864
6	101 HYNERIAN DR		16NOV2017	04JAN2018	15	7	64		BK	44	960
7	11 RYGEL ROAD		02AUG2016	17AUG2017	12	12	96		DP	59	1152
8	121 E. MOYA STREET		06OCT2017	10OCT2017	10	5	40		PUB	38	400
9	1905 DOCK STREET		16OCT2017	20OCT2017	10	5	30		DP	44	300
10	1304 CRESCENT AVE		29JUN2017	30OCT2017	15	5	25		DP	90	375

The syntax of the INPUT function is very similar to the syntax of the PUT function (which performs numeric-to-character conversions).

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

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

However, note that the INPUT function requires an informat, whereas the PUT function requires a format. To remember which function requires a format versus an informat, note that the INPUT function requires an informat.

Automatic Numeric-to-Character Conversion

The automatic conversion of numeric data to character data is very similar to character-to-numeric conversion. Numeric data values are converted to character values whenever they are used in a character context.

For example, the numeric values of the variable Site are converted to character values if you do the following:

- assign the numeric value to a previously defined character variable, such as the character variable SiteCode: SiteCode=site;
- use the numeric value with an operator that requires a character value, such as the concatenation operator: SiteCode=site||dept;
- specify the numeric value in a function that requires character arguments, such as the SUBSTR function: Region=substr(site,1,4);

Specifically, SAS writes the numeric value with the BEST12. format, and the resulting character value is right-aligned. This conversion occurs before the value is assigned or used with any operator or function. However, automatic numeric-to-character conversion can cause unexpected results. For example, suppose the original numeric value has fewer than 12 digits. The resulting character value has leading blanks, which might cause problems when you perform an operation or function.

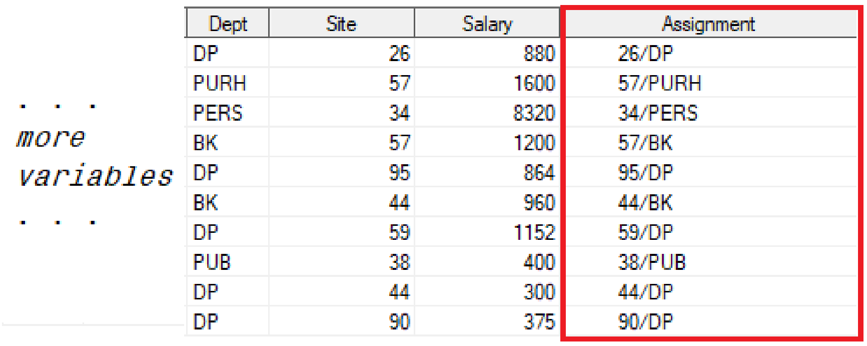
Automatic numeric-to-character conversion also causes a message to be written to the SAS log indicating that the conversion has occurred.

Explicit Numeric-to-Character Conversion

Use the PUT function to explicitly convert numeric data values to character data values.

Suppose you want to create a new character variable named Assignment that concatenates the values of the numeric variable Site and the character variable Dept. The new variable values must contain the value of Site followed by a slash (/) and then the value of Dept (for example, **`26/DP`**).

Figure 14.1 *SAS Data Set Cert.Temp (partial data set)*



. . . *more variables* . . .

Dept	Site	Salary	Assignment
DP	26	880	26/DP
PURH	57	1600	57/PURH
PERS	34	8320	34/PERS
BK	57	1200	57/BK
DP	95	864	95/DP
BK	44	960	44/BK
DP	59	1152	59/DP
PUB	38	400	38/PUB
DP	44	300	44/DP
DP	90	375	90/DP

Here is an assignment statement that contains the concatenation operator (||) to indicate that Site should be concatenated with Dept, using a slash as a separator.

```
data work.newtemp;
  set cert.temp;
  Assignment=site||'/'||dept;
run;
```

Note: The slash is enclosed in quotation marks. All character constants must be enclosed in quotation marks.

Submitting this DATA step causes SAS to automatically convert the numeric values of Site to character values because Site is used in a character context. The variable Site appears with the concatenation operator, which requires character values. To explicitly convert the numeric values of Site to character values, you must add the PUT function to your assignment statement.

Syntax, PUT function:

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

- *source* indicates the numeric variable, constant, or expression to be converted to a character value
- a *format* matching the data type of the source must also be specified, as in this example:

```
put(site,2.)
```

Here are several facts about the PUT function.

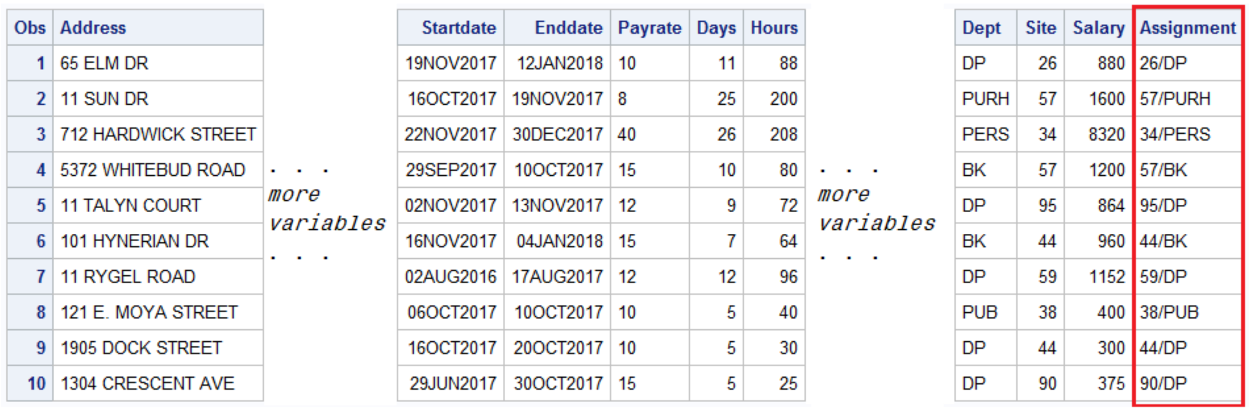
- The PUT function always returns a character string.
- The PUT function returns the source written with a format.
- The format must agree with the source in type.
- Numeric formats right-align the result; character formats left-align the result.
- When you use the PUT function to create a variable that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

When you use a numeric variable as the source, you must specify a numeric format.

To explicitly convert the numeric values of Site to character values, use the PUT function in an assignment statement, where Site is the source variable. Because Site has a length of 2, choose 2. as the numeric format. The DATA step adds the new variable from the assignment statement to the data set.

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

Output 14.2 PROC PRINT Output of Work.NewTemp (partial output)



Obs	Address
1	65 ELM DR
2	11 SUN DR
3	712 HARDWICK STREET
4	5372 WHITEBUD ROAD
5	11 TALYN COURT
6	101 HYNERIAN DR
7	11 RYGEL ROAD
8	121 E. MOYA STREET
9	1905 DOCK STREET
10	1304 CRESCENT AVE

. . . more variables . . .

Startdate	Enddate	Payrate	Days	Hours
19NOV2017	12JAN2018	10	11	88
16OCT2017	19NOV2017	8	25	200
22NOV2017	30DEC2017	40	26	208
29SEP2017	10OCT2017	15	10	80
02NOV2017	13NOV2017	12	9	72
16NOV2017	04JAN2018	15	7	64
02AUG2016	17AUG2017	12	12	96
06OCT2017	10OCT2017	10	5	40
16OCT2017	20OCT2017	10	5	30
29JUN2017	30OCT2017	15	5	25

. . . more variables . . .

Dept	Site	Salary	Assignment
DP	26	880	26/DP
PURH	57	1600	57/PURH
PERS	34	8320	34/PERS
BK	57	1200	57/BK
DP	95	864	95/DP
BK	44	960	44/BK
DP	59	1152	59/DP
PUB	38	400	38/PUB
DP	44	300	44/DP
DP	90	375	90/DP

Notice that no conversion messages appear in the SAS log when you use the PUT function.

Log 14.4 SAS Log

```
9355 data work.newtemp;
9356  set cert.temp;
9357  Assignment=put(site,2.)||'/'||dept;
9358  run;

NOTE: There were 10 observations read from the data set CERT.TEMP.
NOTE: The data set WORK.NEWTEMP has 10 observations and 17 variables.
```

Manipulating SAS Date Values with Functions

SAS Date Functions

SAS stores date, time, and datetime values as numeric values. You can use several functions to create these values. For more information about datetime values, see Chapter 13, “ SAS Date, Time, and Datetime Values,” on page 221.

Table 14.4 Typical Use of SAS Date Functions

Function	Example Code	Result
MDY	`date=mdy(mon,day,yr);`	SAS date

TODAYDATE	`now=today();`
	`now=date();`
		today's date as a SAS date

TIME	`curtime=time();`	current time as a SAS time

Use other functions to extract months, quarters, days, and years from SAS date values.

Table 14.5 Selected Functions to Use with SAS Date Values

Function	Example Code	Result
DAY	`day=day(date);`	day of month (1-31)
QTR	`quarter=qtr(date);`	quarter (1-4)
WEEKDAY	`wkday=weekday(date);`	day of week (1-7)
MONTH	`month=month(date);`	month (1-12)
YEAR	`yr=year(date);`	year (4 digits)
INTCK	`x=intck('day',d1,d2);`	days from D1 to D2
	`x=intck('week',d1,d2);`	weeks from D1 to D2
	`x=intck('month',d1,d2);`	months from D1 to D2
	`x=intck('qtr',d1,d2);`	quarters from D1 to D2
	`x=intck('year',d1,d2);`	years from D1 to D2
INTNX	`x=intnx('interval', start-from,increment);`	date, time, or datetime value
DATDIF	`x=datdif(date1,date2,'ACT/ACT');`	days between date1 and date2
YRDIF	`x=yrdif(date1,date2,'ACT/ACT');`	years between date1 and date2

YEAR, QTR, MONTH, and DAY Functions

Overview of YEAR, QTR, MONTH, and DAY Functions

Every SAS date value can be queried for the values of its year, quarter, month, and day. You extract these values by using the functions YEAR, QTR, MONTH, and DAY. They each work the same way.

Syntax, YEAR, QTR, MONTH, and DAY functions:

YEAR(*date***)**

QTR(*date***)**

MONTH(*date***)**

DAY(*date***)**

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

The YEAR function returns a four-digit numeric value that represents the year (for example, 2018). The QTR function returns a value of 1, 2, 3, or 4 from a SAS date value to indicate the quarter of the year in which a date value falls. The MONTH function returns a numeric value that ranges from 1 to 12, representing the month of the year. The value 1 represents January, 2 represents February, and so on. The DAY function returns a numeric value from 1 to 31, representing the day of the month.

Table 14.6 Selected Date Functions and Their Uses

Function	Description	Form	Sample Value
YEAR	Extracts the year value from a SAS date value.	YEAR*(date)*	`2018`
QTR	Extracts the quarter value from a SAS date value	QTR*(date)*	`1`
MONTH	Extracts the month value from a SAS date value.	MONTH*(date)*	`12`
DAY	Extracts the day value from a SAS date value	DAY*(date)*	`5`

Example: Finding the Year and Month

Suppose you want to create a subset of the data set Cert.Temp that contains information about all temporary employees who were hired in November 2017. The data set Cert.Temp contains the beginning and ending dates for staff employment, but there are no month or year variables in the data set. To determine the year in which employees were hired, you can apply the YEAR function to the variable that contains the employee start date, StartDate. Here is a way to write the YEAR function:

```
year(startdate)
```

Likewise, to determine the month in which employees were hired, you apply the MONTH function to StartDate.

```
month(startdate)
```

To create the new data set, you include these functions in a subsetting IF statement within a DATA step. The subsetting IF statement specifies the new data set include only the observations where the YEAR function extracts a value of 2017 and the MONTH function extracts a value of 11. The value of 11 stands for November.

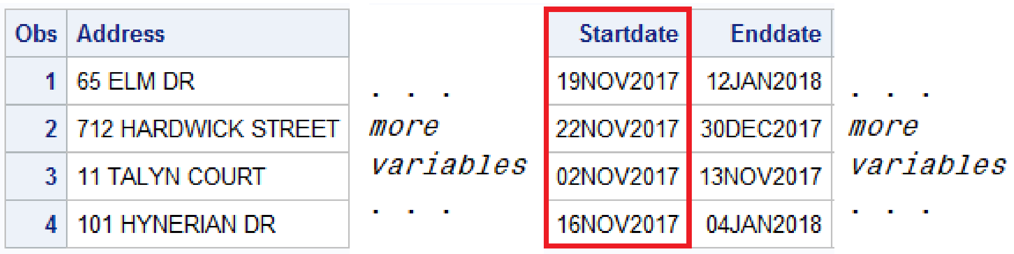
```
data work.nov17;
  set cert.temp;
  if year(startdate)=2017 and month(startdate)=11;
run;
```

When you add a PROC PRINT step to the program, you can view the new data set.

```
proc print data=work.nov17;
  format startdate enddate birthdate date9.;
run;
```

The new data set contains information about only those employees who were hired in November 2017.

Output 14.3 PROC PRINT Output of Work.Nov17 (partial output)



Obs	Address		Startdate	Enddate	
1	65 ELM DR	. . .	19NOV2017	12JAN2018	. . .
2	712 HARDWICK STREET	more	22NOV2017	30DEC2017	more
3	11 TALYN COURT	variables	02NOV2017	13NOV2017	variables
4	101 HYNERIAN DR	. . .	16NOV2017	04JAN2018	. . .

Example: Finding the Year

Suppose you want to create a subset of the data set Cert.Temp that contains information about all temporary employees who were hired during a specific year, such as 2016. Cert.Temp contains the dates on which employees began work with the company and their ending dates, but there is no year variable.

To determine the year in which employees were hired, you can apply the YEAR function to the variable that contains the employee start date, StartDate. You write the YEAR function as follows:

```
year(startdate)
```

To create the new data set, you include this function in a subsetting IF statement within a DATA step. This subsetting IF statement specifies that only observations in which the YEAR function extracts a value of **`2016`** are placed in the new data set.

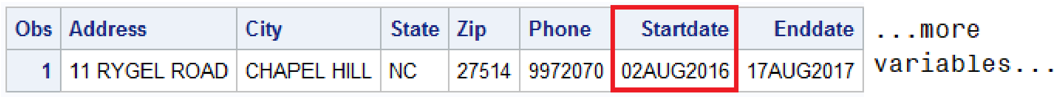
```
data work.temp16;
  set cert.temp;
  if year(startdate)=2016;
run;
```

When you add a PROC PRINT step to the program, you can view the new data set.

```
data work.temp16;
  set cert.temp;
  where year(startdate)=2016;
run;
proc print data=work.temp16;
  format startdate enddate birthdate date9.;
run;
```

The new data set contains information for only those employees who were hired in 2016.

Output 14.4 PROC PRINT Output of Work.Temp16 (partial output)



Obs	Address	City	State	Zip	Phone	Startdate	Enddate	
1	11 RYGEL ROAD	CHAPEL HILL	NC	27514	9972070	02AUG2016	17AUG2017	...more variables...

WEEKDAY Function

Overview of the WEEKDAY Function

The WEEKDAY function enables you to extract the day of the week from a SAS date value.

Syntax, WEEKDAY function:

WEEKDAY(*date*)

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

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

Table 14.7 *Values for the WEEKDAY Function*

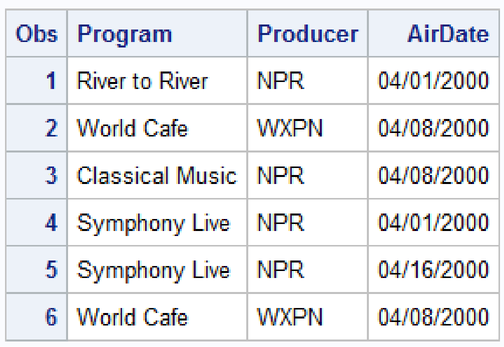
Value	Equals	Day of the Week
1	=	Sunday
2	=	Monday
3	=	Tuesday
4	=	Wednesday
5	=	Thursday
6	=	Friday
7	=	Saturday

Example: WEEKDAY Function

For example, suppose the data set Cert.Sch contains a broadcast schedule. The variable AirDate contains SAS date values. To create a data set that contains only weekend broadcasts, you use the WEEKDAY function in a subsetting IF statement. You include only observations in which the value of AirDate corresponds to a Saturday or Sunday.

```
data work.schwkend;
  set cert.sch;
  if weekday(airdate)in(1,7);
run;
proc print data=work.schwkend;
run;
```

Output 14.5 *PROC PRINT Output of Weekday Function*



Obs	Program	Producer	AirDate
1	River to River	NPR	04/01/2000
2	World Cafe	WXPN	04/08/2000
3	Classical Music	NPR	04/08/2000
4	Symphony Live	NPR	04/01/2000
5	Symphony Live	NPR	04/16/2000
6	World Cafe	WXPN	04/08/2000

Note: In the example above, the statement **`if weekday(airdate) in (1,7);`** is the same as **`if weekday(airdate)=7 or weekday(airdate)=1;`**

MDY Function

Overview of the MDY Function

The MDY function returns a SAS date value from month, day, and year values.

Syntax, MDY function:

MDY (month, day, year)

- *month* specifies a numeric constant, variable, or expression that represents an integer from 1 through 12.
- *day* specifies a numeric constant, variable, or expression that represents an integer from 1 through 31.
- *year* specifies a numeric constant, variable, or expression with a value of a two-digit or four-digit integer that represents that year.

Example: MDY Function

In the data set Cert.Dates, the values for month, day, and year are stored in the numeric variables Month, Day, and Year. It is possible to write the following MDY function to create the SAS date values:

```
mdy(month,day,year)
```

To create a new variable to contain the SAS date values, place this function in an assignment statement.

```
data work.datestemp;
  set cert.dates;
  Date=mdy(month,day,year);
run;
proc print data=work.datestemp;
  format date mmddyy10.;
run;
```

Output 14.6 *PROC PRINT Output of Work.Datestemp*

Obs	year	month	day	date
1	2018	1	22	01/22/2018
2	2018	2	9	02/09/2018
3	2018	3	5	03/05/2018
4	2018	4	27	04/27/2018
5	2018	5	10	05/10/2018
6	2018	6	6	06/06/2018
7	2018	7	23	07/23/2018
8	2018	8	11	08/11/2018
9	2018	9	3	09/03/2018
10	2018	10	5	10/05/2018
11	2018	11	23	11/23/2018
12	2018	12	13	12/13/2018

The MDY function can also add the same SAS date to every observation. This might be useful if you want to compare a fixed beginning date with different end dates. Just use numbers instead of data set variables when providing values to the MDY function.

```
data work.datestemp;
  set cert.dates;
  DateCons=mdy(6,17,2018);
run;
proc print data=work.datestemp;
  format DateCons mmddyy10.;
run;
```

Output 14.7 *PROC PRINT Output of Work.Datestemp*

Obs	year	month	day	DateCons
1	2018	1	22	06/17/2018
2	2018	2	9	06/17/2018
3	2018	3	5	06/17/2018
4	2018	4	27	06/17/2018
5	2018	5	10	06/17/2018
6	2018	6	6	06/17/2018
7	2018	7	23	06/17/2018
8	2018	8	11	06/17/2018
9	2018	9	3	06/17/2018
10	2018	10	5	06/17/2018
11	2018	11	23	06/17/2018
12	2018	12	13	06/17/2018

To display the years clearly, format SAS dates with the DATE9. format. This forces the year to appear with four digits, as shown above in the Date and DateCons variables of the Work.DatesTenp output.

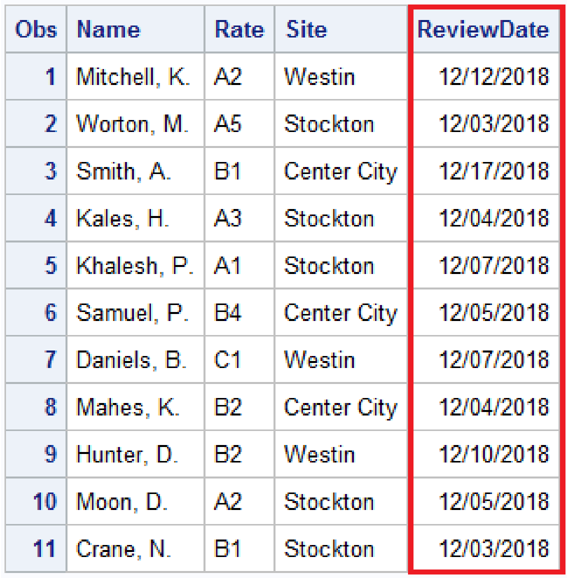
Example: Finding the Date

The data set Cert.Review2018 contains a variable named Day. This variable contains the day of the month for each employee's performance appraisal. The appraisals were all completed in December of 2018.

The following DATA step uses the MDY function to create a new variable named ReviewDate. This variable contains the SAS date value for the date of each performance appraisal.

```
data work.review2018 (drop=Day);
  set cert.review2018;
  ReviewDate=mdy(12,day,2018);
run;
proc print data=work.review2018;
  format ReviewDate mmddyy10.;
run;
```

Output 14.8 *PROC PRINT Output of Work.Review2018*



Obs	Name	Rate	Site	ReviewDate
1	Mitchell, K.	A2	Westin	12/12/2018
2	Worton, M.	A5	Stockton	12/03/2018
3	Smith, A.	B1	Center City	12/17/2018
4	Kales, H.	A3	Stockton	12/04/2018
5	Khalesh, P.	A1	Stockton	12/07/2018
6	Samuel, P.	B4	Center City	12/05/2018
7	Daniels, B.	C1	Westin	12/07/2018
8	Mahes, K.	B2	Center City	12/04/2018
9	Hunter, D.	B2	Westin	12/10/2018
10	Moon, D.	A2	Stockton	12/05/2018
11	Crane, N.	B1	Stockton	12/03/2018

Note: If you specify an invalid date in the MDY function, SAS assigns a missing value to the target variable.

```
data work.review2018 (drop=Day);
  set cert.review2018;
  ReviewDate=mdy(15,day,2018);
run;
proc print data=work.review2018;
  format ReviewDate mmddyy10.;
run;
```

DATE and TODAY Functions

Overview of the DATE Function

The DATE function returns the current date as a numeric SAS date value.

Note: If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the return values for date and time are determined by the time zone.

Syntax, DATE function:

DATE ()

The DATE function does not require any arguments, but it must be followed by parentheses.

The DATE function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

Overview of the TODAY Function

The TODAY function returns the current date as a numeric SAS date value.

Note: If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the return values for date and time are determined by the time zone.

Syntax, TODAY function:

TODAY ()

The TODAY function does not require any arguments, but it must be followed by parentheses.

The TODAY function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

Example: The DATE and TODAY Functions

The DATE and TODAY functions have the same form and can be used interchangeably. To add a new variable, which contains the current date, to the data set Cert.Temp. To create this variable, write an assignment statement such as the following:

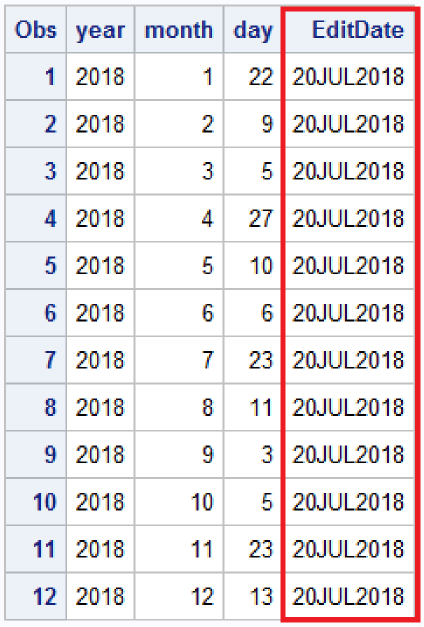
```
EditDate=date();
```

After this statement is added to a DATA step and the step is submitted, the data set that contains EditDate is created. To display these SAS date values in a different form, you can associate a SAS format with the values. For example, the FORMAT statement below associates the DATE9. format with the variable EditDate. The output that is created by this PROC PRINT step appears below.

Note: For this example, the SAS date values shown below were created by submitting this program on July 20, 2018.

```
data work.tempdate;
  set cert.dates;
  EditDate=date();
run;
proc print data=work.tempdate;
  format EditDate date9.;
run;
```

Output 14.9 *PROC PRINT Output of Work.TempDate*



Obs	year	month	day	EditDate
1	2018	1	22	20JUL2018
2	2018	2	9	20JUL2018
3	2018	3	5	20JUL2018
4	2018	4	27	20JUL2018
5	2018	5	10	20JUL2018
6	2018	6	6	20JUL2018
7	2018	7	23	20JUL2018
8	2018	8	11	20JUL2018
9	2018	9	3	20JUL2018
10	2018	10	5	20JUL2018
11	2018	11	23	20JUL2018
12	2018	12	13	20JUL2018

INTCK Function

Overview of the INTCK Function

The INTCK function returns the number of interval boundaries of a given kind that lie between two dates, times, or datetime values. You can use it to count the passage of days, weeks, months, and so on.

Syntax, INTCK function:

INTCK (‘*interval*’ , *from*, *to*)

- ‘*interval*’ specifies a character constant or a variable. Interval can appear in uppercase or lowercase. The value can be one of the following:
  - DAY
  - WEEKDAY
  - WEEK
  - TENDAY
  - SEMIMONTH
  - MONTH
  - QTR
  - SEMIYEAR
  - YEAR
- *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.

Note: The type of interval (date, time, or datetime) must match the type of value in *from*.

Details

The INTCK function 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, and YEAR intervals are counted from 01JAN, not in 365-day multiples.

Consider the results in the following table. The values that are assigned to the variables Weeks, Months, and Years are based on consecutive days.

Table 14.8 *Examples of SAS Statements and Their Values*

Example Code	Value
`Weeks=intck('week','31dec2017'd,'01jan2018'd);`	0
`Months=intck('month','31dec2017'd,'01jan2018'd);`	1
`Years=intck('year','31dec2017'd,'01jan2018'd);`	1

Because December 31, 2017, is a Sunday, no WEEK interval is crossed between that day and January 1, 2018. However, both MONTH and YEAR intervals are crossed.

Examples: INTCK Function

The following statement creates the variable Years and assigns it a value of **2**. The INTCK function determines that two years have elapsed between June 15, 2016, and June 15, 2018.

```
Years=intck('year','15jun2016'd,'15jun2018'd);
```

Note: As shown here, the *from* and *to* dates are often specified as date constants.

Likewise, the following statement assigns the value 24 to the variable Months.

```
Months=intck('month','15jun2016'd,'15jun2018'd);
```

However, the following statement assigns 0 to the variable Years, even though 364 days have elapsed. In this case, the YEAR boundary (01JAN) is not crossed.

```
Years=intck('year','01jan2018'd,'31dec2018'd);
```

Example: The INTCK Function and Periodic Events

A common use of the INTCK function is to identify periodic events such as due dates and anniversaries.

The following 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.

```
data work.anniversary;
  set cert.mechanics(keep=id lastname firstname hired);
  Years=intck('year',hired,today());
  if years=20 and month(hired)=month(today());
run;
proc print data=work.anniversary;
  title '20-Year Anniversaries';
run;
```

The following output is created when the program is run in July 2018.

Output 14.10 *PROC PRINT Output of Work.Anniversary*

20 Year Anniversaries This Month

Obs	ID	LastName	FirstName	Hired	years
1	1499	BAREFOOT	JOSEPH	23JUL98	20
2	1065	CHAPMAN	NEIL	23JUL98	20
3	1406	FOSTER	GERALD	10JUL98	20
4	1423	OSWALD	LESLIE	16JUL98	20

INTNX Function

Overview of the INTNX Function

The INTNX function is similar to the INTCK function. The INTNX function applies multiples of a given interval to a date, time, or datetime value and returns the resulting value. You can use the INTNX function to identify past or future days, weeks, months, and so on.

Syntax, INTNX function:

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

- *'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.

Note: The type of interval (date, time, or datetime) must match the type of value in *start-from* and *increment*.

Details

When you specify date intervals, the value of the character constant or variable that is used in *interval* can be one of the following:

- DATETIME
- DAY
- QTR
- MONTH
- SEMIMONTH
- SEMIYEAR
- TENDAY
- TIME
- WEEK
- WEEKDAY
- YEAR

Example: INTNX Function

For example, the following statement creates the variable TargetYear and assigns it a SAS date value of **22281**, which corresponds to January 1, 2021.

```
TargetYear=intnx('year','20Jul18'd,3);
```

Likewise, the following statement assigns the value for the date July 1, 2018, to the variable TargetMonth.

```
TargetMonth=intnx('semiyear','01Jan18'd,1);
```

SAS date values are based on the number of days since January 1, 1960. Yet the INTNX function can use intervals of weeks, months, years, and so on.

The purpose of the optional alignment argument is to specify whether the returned value should be at the beginning, middle, or end of the interval. When specifying date alignment in the INTNX function, use the following values or their corresponding aliases:

- BEGINNING Alias: B
- MIDDLE Alias: M
- END Alias: E
- SAME Alias: SAMEDAY or S

The best way to understand the alignment argument is to see its effect on identical statements. The following table shows the results of three INTNX statements that differ only in the value of alignment.

Table 14.9 *Alignment Values for the INTNX Function*

Example Code	**Date Value**
`MonthX=intnx('month','01jan2018'd,5,'b');`	**21336** (June 1, 2018)
`MonthX=intnx('month','01jan2018'd,5,'m');`	**21350** (June 15, 2018)
`MonthX=intnx('month','01jan2018'd,5,'e');`	**21365** (June 30, 2018)

These INTNX statements 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.

DATDIF and YRDIF Functions

The DATDIF and YRDIF functions calculate the difference in days and years between two SAS dates, respectively. Both functions accept start dates and end dates that are specified as SAS date values. Also, both functions use a basis argument that describes how SAS calculates the date difference.

Syntax, DATDIF, and YRDIF functions:

DATDIF(*start_date,end_date,basis)***)**

YRDIF(*start_date,end_date,basis)***)**

- *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.

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.

Table 14.10 *Character Strings in the DATDIF Function*

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

The best way to understand the different options for the basis argument is to see the different effects that they have on the value that the function returns. The table below lists four YRDIF functions that use the same start date and end date. Each function uses one of the possible values for basis, and each one returns a different value.

Table 14.11 *Examples of the YRDIF Function*

Example Code	Returned Value

`data _null_;`
`  x=yrdif('16feb2016'd,'16jun2018'd,'30/360');`
`  put x;`
`run;`
	`2.3333333333`

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/ACT');`
`  put x;`
`run;`
	`2.3291114604`

Example Code	Returned Value

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/360');`
`  put x;`
`run;`
	2.3638888889

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/365');`
`  put x;`
`run;`
	2.3315068493

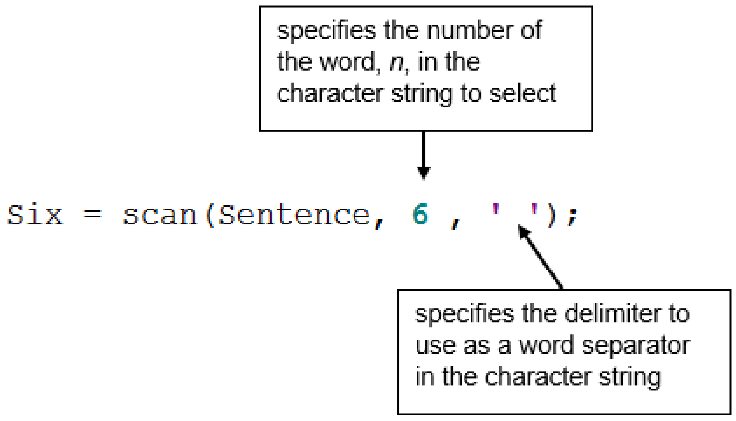
Modifying Character Values with Functions

SCAN Function

Overview of the SCAN Function

The SCAN function returns the *n*th word from a character string. The SCAN function enables you to separate a character value into words and to return a specified word.

Figure 14.2 *SCAN Function*


specifies the number of the word, *n*, in the character string to select
Six = scan(Sentence, 6 , ' ');
specifies the delimiter to use as a word separator in the character string

Note: In SAS 9.4 or later, if the variable has not yet been given a length, then the SCAN function returns the value and assigns the variable the given length of the first argument. In SAS 9.3 or earlier, by default, the variable is assigned a length of 200.

Syntax, SCAN function:

SCAN(*argument,n<,<delimiters>>***)**

- *argument* specifies the character variable or expression to scan.
- *n* specifies which word to return.
- *delimiters* are special characters that must be enclosed in single quotation marks (' '). If you do not specify *delimiters*, default delimiters are used.

Details

- Leading delimiters before the first word in the character string do not affect the SCAN function.
- If there are two or more contiguous delimiters, the SCAN function treats them as one.
- If *n* is greater than the number of words in the character string, the SCAN function returns a blank value.
- If *n* is negative, the SCAN function selects the word in the character string starting from the end of the string.

Example: Create New Name Variables

Use the SCAN function to create your new name variables for Cert.Staff. First, examine the values of the existing Name variable to determine which characters separate the names in the values.

```
LastName=scan(name,1);
```

Notice that blanks and commas appear between the names and that the employee's last name appears first, then the first name, and then the middle name.

To create the LastName variable to store the employee's last name, you write an assignment statement that contains the following SCAN function:

```
LastName=scan(name,1,' ,');
```

Note that a blank and a comma are specified as delimiters. You can also write the function without listing delimiters, because the blank and comma are default delimiters.

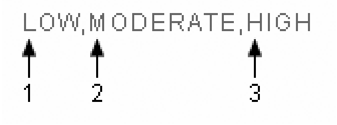
```
LastName=scan(name,1);
```

Here is the complete DATA step that is needed to create LastName, FirstName, and MiddleName. Notice that the original Name variable is dropped from the new data set.

```
data work.newnames(drop=name);
  set cert.staff;
  LastName=scan(name,1);
  FirstName=scan(name,2);
run;
```

Specifying Delimiters

The SCAN function uses delimiters to separate a character string into words. Delimiters are characters that are specified as word separators. For example, if you are working with the character string below and you specify the comma as a delimiter, the SCAN function separates the string into three words.



Then the function returns the word that you specify. In this example, if you specify the third word, the SCAN function returns the word HIGH.

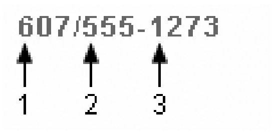
Here is another example that uses the comma as a delimiter, and specifies that the third word be returned.

```
    209 RADCLIFFE ROAD, CENTER CITY, NY, 92716
```

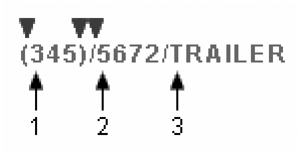
In this example, if you specify the third word, the word returned by the SCAN function is NY (NY contains a leading blank).

Specifying Multiple Delimiters

When using the SCAN function, you can specify as many delimiters as needed to correctly separate the character expression. When you specify multiple delimiters, SAS uses any of the delimiters, singly or in any combination, as word separators. For example, if you specify both the slash and the hyphen as delimiters, the SCAN function separates the following text string into three words:



The SCAN function treats two or more contiguous delimiters, such as the parenthesis and slash below, as one delimiter. Also, leading delimiters have no effect.



Default Delimiters

If you do not specify delimiters when using the SCAN function, default delimiters are used. Here are the default delimiters:

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

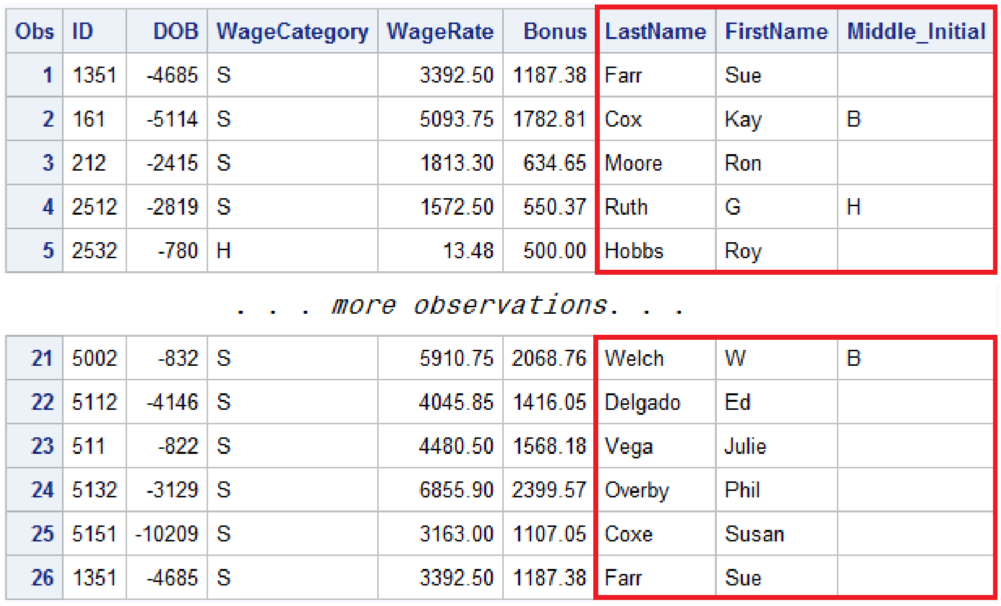
Specifying Variable Length

If a variable is not assigned a length before it is specified in the SCAN function, the variable is given the length of the first argument. This length could be too small or too large for the remaining variables.

You can add a LENGTH statement to the DATA step, and specify an appropriate length for all three variables. The LENGTH statement is placed before the assignment statement that contains the SCAN function so that SAS can specify the length the first time it encounters the variable.

```
data work.newnames(drop=name);
  set cert.staff;
  length LastName FirstName $ 12;
  LastName=scan(name,1);
  FirstName=scan(name,2);
  MiddleInitial=scan(name,3);
run;
proc print data=newnames;
run;
```

Output 14.11 *PROC PRINT Output of Work.NewNames (partial output)*



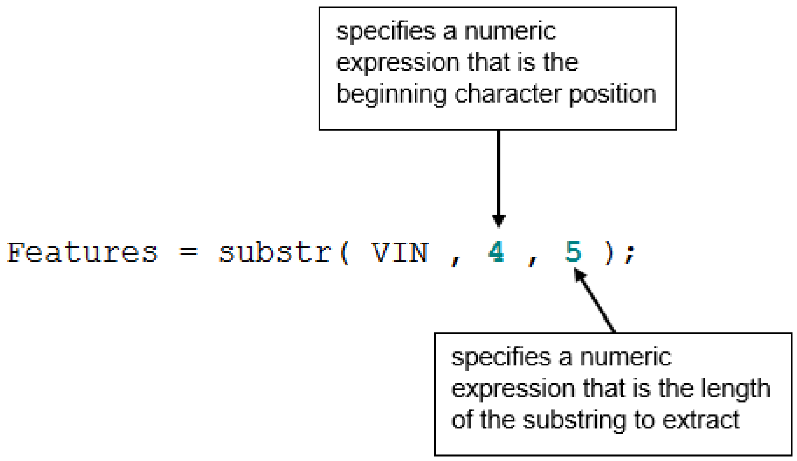
Obs	ID	DOB	WageCategory	WageRate	Bonus	LastName	FirstName	Middle_Initial
1	1351	-4685	S	3392.50	1187.38	Farr	Sue	
2	161	-5114	S	5093.75	1782.81	Cox	Kay	B
3	212	-2415	S	1813.30	634.65	Moore	Ron	
4	2512	-2819	S	1572.50	550.37	Ruth	G	H
5	2532	-780	H	13.48	500.00	Hobbs	Roy	
. . . more observations. . .								
21	5002	-832	S	5910.75	2068.76	Welch	W	B
22	5112	-4146	S	4045.85	1416.05	Delgado	Ed	
23	511	-822	S	4480.50	1568.18	Vega	Julie	
24	5132	-3129	S	6855.90	2399.57	Overby	Phil	
25	5151	-10209	S	3163.00	1107.05	Coxe	Susan	
26	1351	-4685	S	3392.50	1187.38	Farr	Sue	

SUBSTR Function

Overview of the SUBSTR Function

The SUBSTR function extracts a substring from an argument, starting at a specific position in the string.

Figure 14.3 *SUBSTR Function*



```
Features = substr( VIN , 4 , 5 );
```

specifies a numeric expression that is the beginning character position
specifies a numeric expression that is the length of the substring to extract

The SUBSTR function can be used on either the right or left of the equal sign to replace character value constants.

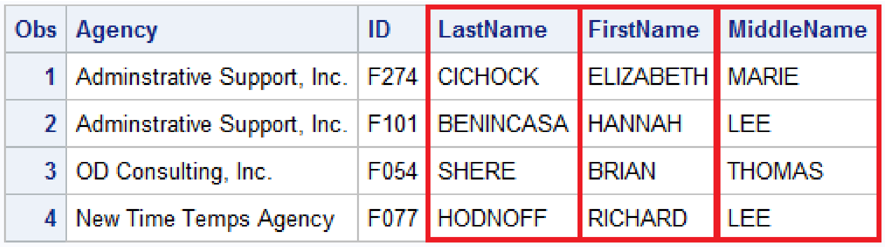
Syntax, SUBSTR function:

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

- *argument* specifies the character variable or expression to scan.
- *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.

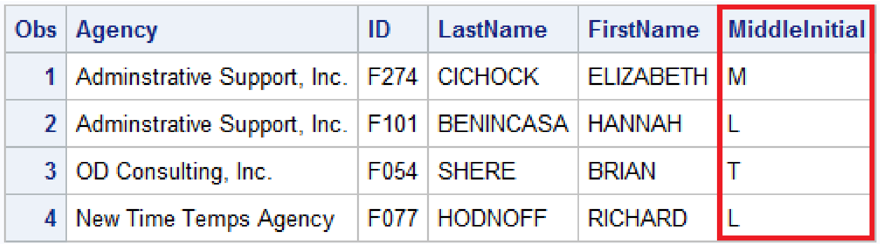
Example: SUBSTR Function

This example begins with the task of extracting a portion of a value. In the data set Cert.AgencyEmp, the names of temporary employees are stored in three name variables: LastName, FirstName, and MiddleName.



Obs	Agency	ID	LastName	FirstName	MiddleName
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	MARIE
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	LEE
3	OD Consulting, Inc.	F054	SHERE	BRIAN	THOMAS
4	New Time Temps Agency	F077	HODNOFF	RICHARD	LEE

However, suppose you want to modify the data set to store only the middle initial instead of the full middle name. To do so, you must extract the first letter of the middle name values and assign these values to the new variable MiddleInitial.



Obs	Agency	ID	LastName	FirstName	MiddleInitial
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	M
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	L
3	OD Consulting, Inc.	F054	SHERE	BRIAN	T
4	New Time Temps Agency	F077	HODNOFF	RICHARD	L

Using the SUBSTR function, you can extract the first letter of the MiddleName value to create the new variable MiddleInitial.

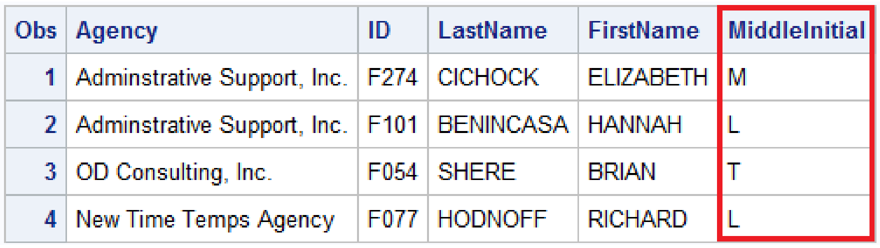
You write the SUBSTR function as the following:

```
substr(middlename,1,1)
```

This function extracts a character string from the value of MiddleName. The string to be extracted begins in position 1 and contains one character. This function is placed in an assignment statement in the DATA step.

```
data work.agencyemp(drop=middlename);
  set cert.agencyemp;
  length MiddleInitial $ 1;
  MiddleInitial=substr(middlename,1,1);
run;
proc print data=work.agencyemp;
run;
```

The new MiddleInitial variable is given the same length as MiddleName. The MiddleName variable is then dropped from the new data set.

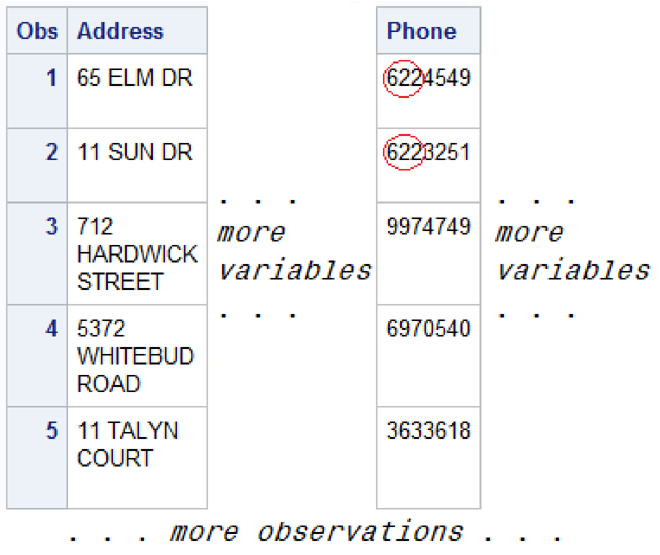


Obs	Agency	ID	LastName	FirstName	MiddleInitial
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	M
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	L
3	OD Consulting, Inc.	F054	SHERE	BRIAN	T
4	New Time Temps Agency	F077	HODNOFF	RICHARD	L

You can use the SUBSTR function to extract a substring from any character value if you know the position of the value.

Replacing Text Using SUBSTR

There is a second use for the SUBSTR function. This function can also be used to replace the contents of a character variable. For example, suppose the local phone exchange `622` was replaced by the exchange `433`. You need to update the character variable Phone in Cert.Temp to reflect this change.



Obs	Address	. . . more variables . . .	Phone	. . . more variables . . .
1	65 ELM DR		6224549	
2	11 SUN DR		6223251	
3	712 HARDWICK STREET		9974749	
4	5372 WHITEBUD ROAD		6970540	
5	11 TALYN COURT		3633618	

. . . more observations . . .

You can use the SUBSTR function to complete this modification. The syntax of the SUBSTR function, when used to replace a variable's values, is identical to the syntax for extracting a substring.

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

However, in this case, note the following:

- The first argument specifies the character variable whose values are to be modified.
- The second argument specifies the position at which the replacement is to begin.
- The third argument specifies the number of characters to replace. If *n* is omitted, all remaining characters are replaced.

Positioning the SUBSTR Function

SAS uses the SUBSTR function to extract a substring or to modify a variable's values, depending on the position of the function in the assignment statement.

When the function is on the right side of an assignment statement, the function returns the requested string.

```
MiddleInitial=substr(middlename,1,1);
```

But if you place the SUBSTR function on the left side of an assignment statement, the function is used to modify variable values.

```
substr(region,1,3)='NNW';
```

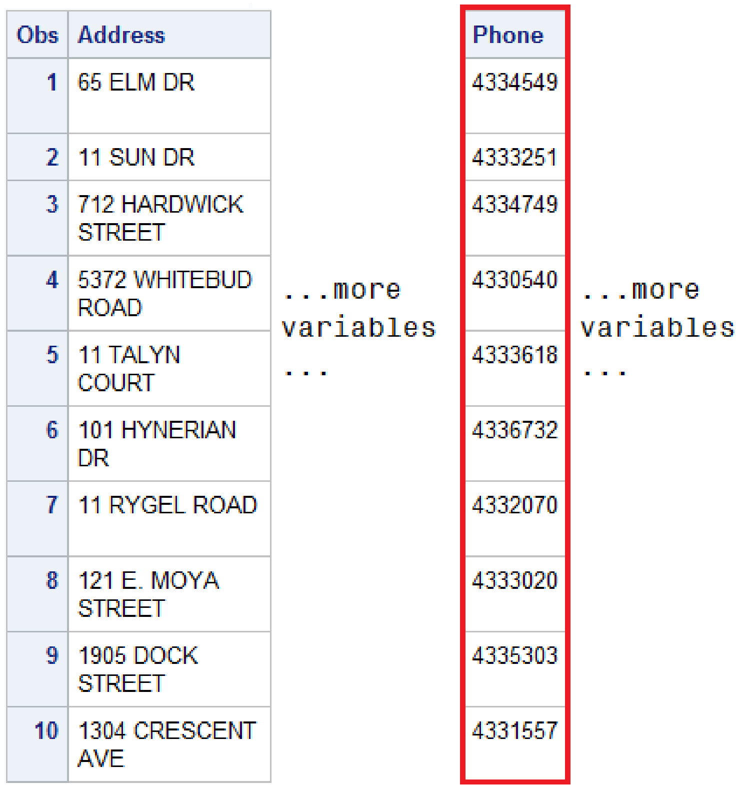
When the SUBSTR function modifies variable values, the right side of the assignment statement must specify the value to place into the variable. For example, to replace the fourth and fifth characters of a variable named Test with the value **92**, you write the following assignment statement:

```
substr(test,4,2)='92';

Test              Test

S7381K2    →    S7392K2
S7381K7    →    S7392K7
```

It is possible to use the SUBSTR function to replace the **622** exchange in the variable Phone. This assignment statement specifies that the new exchange **433** should be placed in the variable Phone, starting at character position 1 and replacing three characters.

```
data work.temp2;
  set cert.temp;
  substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

However, executing this DATA step places the value **433** into all values of Phone.



Obs	Address	...more variables...	Phone	...more variables...
1	65 ELM DR		4334549	
2	11 SUN DR		4333251	
3	712 HARDWICK STREET		4334749	
4	5372 WHITEBUD ROAD		4330540	
5	11 TALYN COURT		4333618	
6	101 HYNERIAN DR		4336732	
7	11 RYGEL ROAD		4332070	
8	121 E. MOYA STREET		4333020	
9	1905 DOCK STREET		4335303	
10	1304 CRESCENT AVE		4331557	

You need to replace only the values of Phone that contain the **622** exchange. To extract the exchange from Phone, add an assignment statement to the DATA step. Notice that the SUBSTR function is used on the right side of the assignment statement.

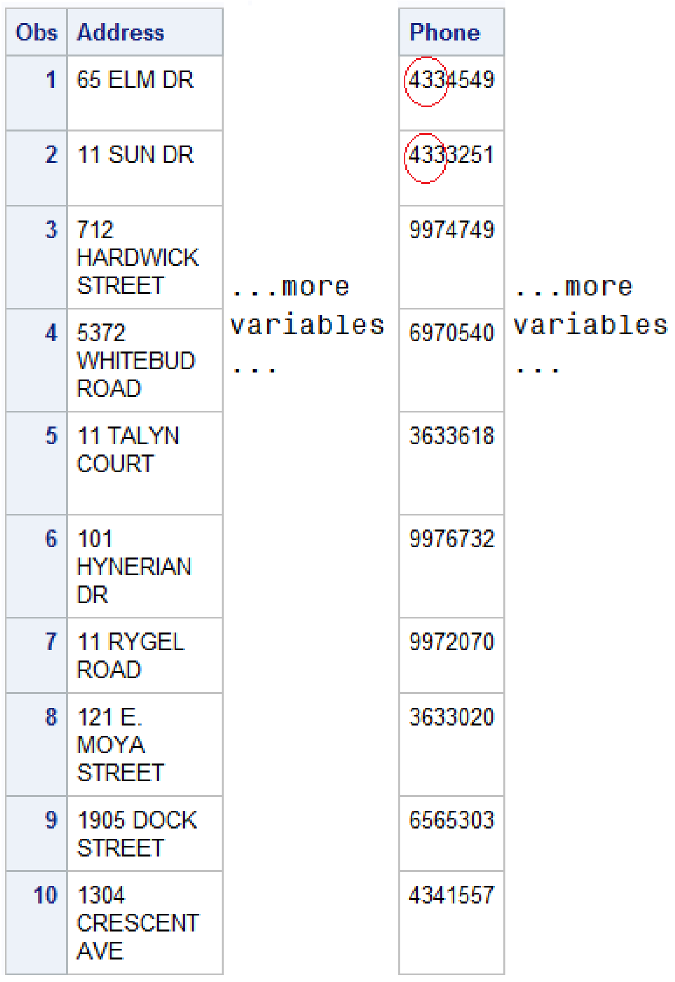
```
data work.temp2(drop=exchange);
  set cert.temp;
  Exchange=substr(phone,1,3);
  substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

Now the DATA step needs an IF-THEN statement to verify the value of the variable Exchange. If the exchange is **622**, the assignment statement executes to replace the value of Phone.

```
data work.temp2(drop=exchange);
  set cert.temp;
  Exchange=substr(phone,1,3);
  if exchange='622' then substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

After the DATA step is executed, the appropriate values of Phone contain the new exchange.

Figure 14.4 *PROC PRINT Output of Work.Temp2 (partial output)*



Obs	Address		Phone	
1	65 ELM DR		4334549	
2	11 SUN DR		4333251	
3	712 HARDWICK STREET		9974749	
4	5372 WHITEBUD ROAD	...more variables...	6970540	...more variables...
5	11 TALYN COURT		3633618	
6	101 HYNERIAN DR		9976732	
7	11 RYGEL ROAD		9972070	
8	121 E. MOYA STREET		3633020	
9	1905 DOCK STREET		6565303	
10	1304 CRESCENT AVE		4341557	

To summarize, when the SUBSTR function is on the right side of an assignment statement, the function extracts a substring.

```
MiddleInitial=substr(middlename,1,1);
```

When the SUBSTR function is on the left side of an assignment statement, the function replaces the contents of a character variable.

```
 substr(region,1,3)='NNW';
```

SCAN versus SUBSTR Functions

The SUBSTR function is similar to the SCAN function. Here is a brief comparison. Both the SCAN and SUBSTR functions can extract a substring from a character value:

- SCAN extracts words within a value that is marked by delimiters.
- SUBSTR extracts a portion of a value by starting at a specified location.

The SUBSTR function is best used when you know the exact position of the string that you want to extract from the character value. It is unnecessary to mark the string by delimiters. For example, the first two characters of the variable ID identify the class level of college students. The position of these characters does not vary within the values of ID.

The SUBSTR function is the best choice to extract class level information from ID. By contrast, the SCAN function is best used during the following actions:

- You know the order of the words in the character value.
- The starting position of the words varies.
- The words are marked by some delimiter.

LEFT and RIGHT Functions

Overview of the 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 and RIGHT function:

LEFT(*argument***)**

RIGHT(*argument***)**

argument specifies a character constant, variable, or expression.

Example: LEFT Function

The following example uses the LEFT function to left-align character expressions.

```
data _null_;
  a='DUE DATE';
  b='   DUE DATE';
  c=left(a);
  d=left(b);
  put c $8.;
  put d $12.;
run;
```

The following is displayed in the SAS log. The LEFT function returns the argument with leading blanks moved to the end of the value. In the example, b has three leading blanks and, in the output, the leading blanks are moved to the end of DUE DATE. DUE DATE is left-aligned.

```
DUE DATE
DUE DATE
```

Example: RIGHT Function

The following example uses the RIGHT function to right-align character expressions.

```
data _null_;
  a='DUE DATE';
  b='DUE DATE   ';
  c=right(a);
  d=right(b);
  put c $8.;
  put d $12.;
run;
```

The following is displayed in the SAS log. The RIGHT function returns the argument with leading blanks moved to the front of the value. In the example, b has three trailing blanks and, in the output, the trailing blanks are moved before DUE DATE. DUE DATE is right-aligned.

```
DUE DATE
   DUE DATE
```

Concatenation Operator

The concatenation operator concatenates character values. The operator can be expressed as || (two vertical bars), ¦¦ (two broken vertical bars), or !!(two exclamation points).

```
FullName = First || Middle || Last;
```

The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation. You can also 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.

TRIM Function

Overview of the TRIM Function

The TRIM function removes trailing blanks from character expressions and returns one blank if the expression contains missing values.

```
FullName = trim(First) || trim(Middle) || Last;
```

The TRIM function is useful for concatenating because the concatenation operator does not remove trailing blanks.

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

Syntax, TRIM function:

TRIM(*argument*)

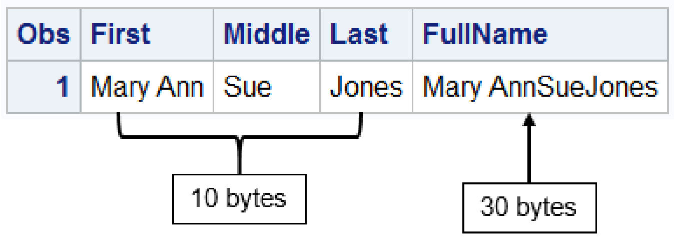
argument can be any character expression. Here are examples:

- a character variable: trim(address)
- another character function: trim(left(id))

Example: TRIM Function

```
data work.nametrim;
  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 = trim(First) || trim(Middle) ||Last;
  drop Name;
run;

proc print data=work.nametrim;
run;
```

Figure 14.5 TRIM Function



Obs	First	Middle	Last	FullName
1	Mary Ann	Sue	Jones	Mary AnnSueJones

10 bytes
30 bytes

CATX Function

Overview of the CATX Function

The CATX function enables you to concatenate character strings, remove leading and trailing blanks, and insert separators. The CATX function returns a value to a variable, or returns a value to a temporary buffer. The 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.

In the DATA step, if the CATX function returns a value to a variable that has not previously been assigned a length, then the variable is given the length of 200. To save storage space, you can add a LENGTH statement to your DATA step, and specify an appropriate length for your variable. The LENGTH statement is placed before the assignment statement that contains the CATX function so that SAS can specify the length the first time it encounters the variable.

If the variable has not previously been assigned a length, the concatenation operator (||) returns a value to a variable. The variable's given length is the sum of the length of the values that are being concatenated. Otherwise, you can use the LENGTH statement before the assignment statement containing the TRIM function to assign a length.

Recall that you can use the TRIM function with the concatenation operator to create one address variable. The address variable contains the values of the three variables Address, City, and Zip. To remove extra blanks from the new values, use the DATA step shown below:

```
data work.newaddress(drop=address city state zip);
  set cert.temp;
  NewAddress=trim(address)||', '||trim(city)||', '||zip;
run;
```

You can accomplish the same concatenation using only the CATX function.

Syntax, CATX function:

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

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

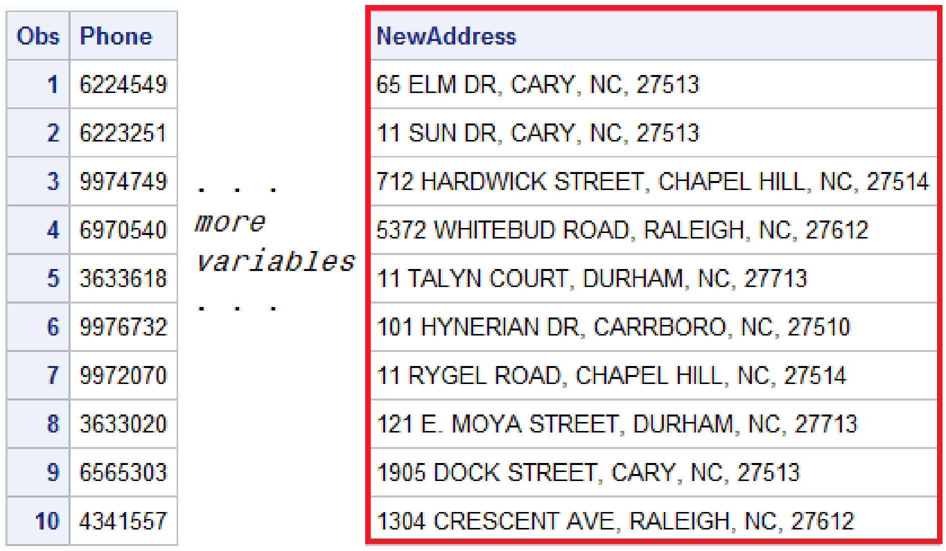
Example: Create New Variable Using CATX Function

You want to create the new variable NewAddress by concatenating the values of the Address, City, and Zip variables from the data set Cert.Temp. You want to strip excess blanks from the old variable's values and separate the variable values with a comma and a space. The DATA step below uses the CATX function to create NewAddress.

```
data work.newaddress(drop=address city state zip);
  set cert.temp;
  NewAddress=catx(', ',address,city,zip);
run;
proc print data=work.newaddress;
run;
```

The revised DATA step creates the values that you would expect for NewAddress.

Output 14.12 *SAS Data Set Work.NewAddress*



Obs	Phone		NewAddress
1	6224549		65 ELM DR, CARY, NC, 27513
2	6223251		11 SUN DR, CARY, NC, 27513
3	9974749	. . .	712 HARDWICK STREET, CHAPEL HILL, NC, 27514
4	6970540	more	5372 WHITEBUD ROAD, RALEIGH, NC, 27612
5	3633618	variables	11 TALYN COURT, DURHAM, NC, 27713
6	9976732	. . .	101 HYNERIAN DR, CARRBORO, NC, 27510
7	9972070		11 RYGEL ROAD, CHAPEL HILL, NC, 27514
8	3633020		121 E. MOYA STREET, DURHAM, NC, 27713
9	6565303		1905 DOCK STREET, CARY, NC, 27513
10	4341557		1304 CRESCENT AVE, RALEIGH, NC, 27612

INDEX Function

Overview of the INDEX Function

The INDEX function enables you to search a character value for a specified string. The INDEX function searches values from left to right, looking for the first occurrence of the string. It returns the position of the string's first character. If the string is not found, it returns a value of 0.

Syntax, INDEX function:

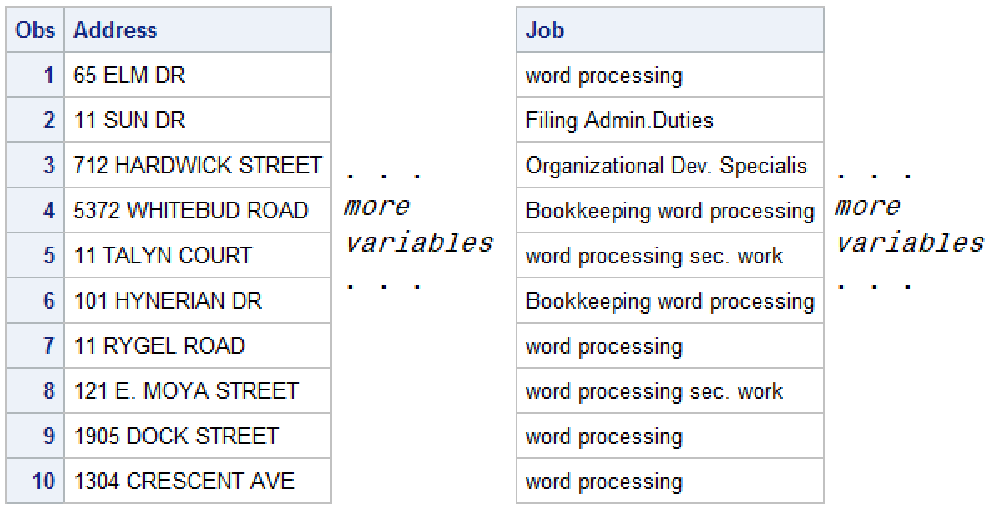
INDEX(*source, excerpt***)**

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

Example: Search for Occurrences of a Phrase

Suppose you want to search the values of the variable Job, which lists job skills. You want to create a data set that contains the names of all temporary employees who have word processing experience. The following figure shows a partial output of the Cert.Temp data set.

Figure 14.6 *Cert.Temp (partial output)*



Obs	Address
1	65 ELM DR
2	11 SUN DR
3	712 HARDWICK STREET
4	5372 WHITEBUD ROAD
5	11 TALYN COURT
6	101 HYNERIAN DR
7	11 RYGEL ROAD
8	121 E. MOYA STREET
9	1905 DOCK STREET
10	1304 CRESCENT AVE

. . . more variables . . .

Job
word processing
Filing Admin.Duties
Organizational Dev. Specialis
Bookkeeping word processing
word processing sec. work
Bookkeeping word processing
word processing
word processing sec. work
word processing
word processing

. . . more variables . . .

To search for the occurrences of the phrase “word processing” in the values of the variable Job, you write the INDEX function as shown below. Note that the character string is enclosed in quotation marks.

```
index(job,'word processing')
```

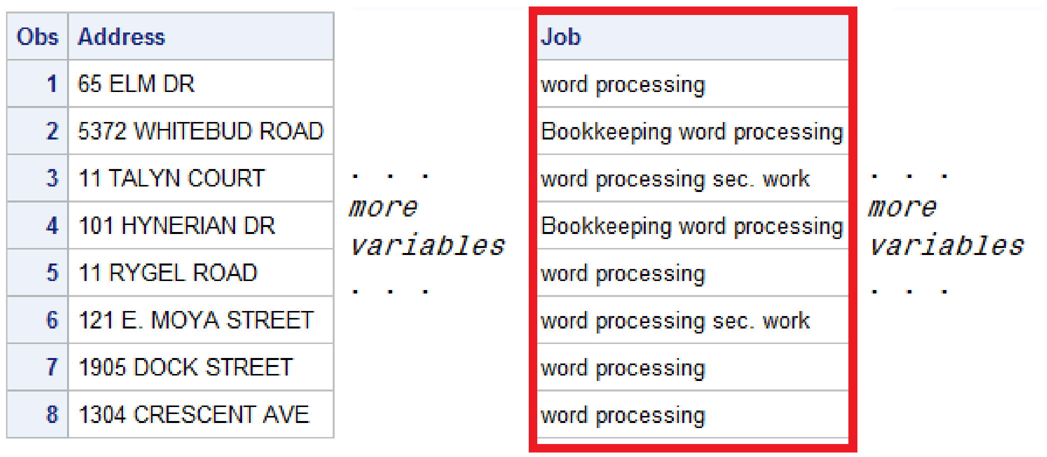
To create the new data set, include the INDEX function in a subsetting IF statement. Only those observations in which the function locates the string and returns a value greater than 0 are written to the data set.

```
data work.datapool;
  set cert.temp;
  where index(job,'word processing') > 0;
run;
```

```
proc print data=work.datapool;
run;
```

Here is the data set that shows the temporary employees who have word processing experience. The program processed all of the observations in the Cert.Temp data set.

Output 14.13 *Work.DataPool (partial output)*



Obs	Address		Job	
1	65 ELM DR		word processing	
2	5372 WHITEBUD ROAD		Bookkeeping word processing	
3	11 TALYN COURT	. . .	word processing sec. work	. . .
4	101 HYNERIAN DR	more variables	Bookkeeping word processing	more variables
5	11 RYGEL ROAD	. . .	word processing	. . .
6	121 E. MOYA STREET		word processing sec. work	
7	1905 DOCK STREET		word processing	
8	1304 CRESCENT AVE		word processing	

Note that the INDEX function is case sensitive, so the character string that you search for must be specified exactly as it is recorded in the data set. For example, the INDEX function shown below would not locate any employees who have word-processing experience.

```
index(job,'WORD PROCESSING')
```

Finding a String Regardless of Case

To ensure that all occurrences of a character string are found, you can use the UPCASE or LOWCASE function with the INDEX function. The UPCASE and LOWCASE functions enable you to convert variable values to uppercase or lowercase letters. You can then specify the character string in the INDEX function accordingly.

```
index(upcase(job),'WORD PROCESSING')

index(lowcase(job),'word processing')
```

FIND Function

Overview of the FIND Function

The FIND function enables you to search for a specific substring of characters within a specified character string.

- The FIND function searches the string, from left to right, for the first occurrence of the substring, and returns the position in the string of the substring's first character.
- If the substring is not found in the string, the FIND function returns a value of 0.
- If there are multiple occurrences of the substring, the FIND function returns only the position of the first occurrence.

Syntax, FIND function:

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

- *string* specifies a character constant, variable, or expression that is searched for substrings.
- *substring* is a character constant, variable, or expression that specifies the substring of characters to search for in *string*.
- *modifiers* is a character constant, variable, or expression that specifies one or more modifiers.
- *startpos* is an integer that specifies the position at which the search should start and the direction of the search. The default value for *startpos* is 1.

Note: If *string* or *substring* is a character literal, you must enclose it in quotation marks.

Details

The modifiers argument enables you to specify 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.

Here are several facts about modifiers and constants.

- If the modifier is a constant, enclose it in quotation marks.
- Specify multiple constants in a single set of quotation marks.
- Modifier values are not case sensitive.

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 specified, the absolute value of startpos determines the position at which to start the search. The sign of startpos determines the direction of the search. That is, when startpos is positive, FIND searches from startpos to the right, When startpos is negative, FIND searches from startpos to the left.

Example: Find Word Processing Jobs in a Data Set

The values of the variable Job are all lowercase. Therefore, to search for the occurrence of word processing in the values of the variable Job, you write the FIND function as shown below. Note that the character substring is enclosed in quotation marks.

```
find(job,'word processing')
```

To create the new data set, include the FIND function in a subsetting IF statement. Only those observations in which the function locates the string and returns a value greater than 0 are written to the data set.

```
data work.datapool;
  set cert.temp;
  where find(job,'word processing') > 0;
run;
proc print data=work.datapool;
run;
```

Output 14.14 *Work.DataPool (partial output)*

Obs	Address	. . . more variables . . .	Job	. . . more variables . . .
1	65 ELM DR		word processing	
2	5372 WHITEBUD ROAD		Bookkeeping word processing	
3	11 TALYN COURT		word processing sec. work	
4	101 HYNERIAN DR		Bookkeeping word processing	
5	11 RYGEL ROAD		word processing	
6	121 E. MOYA STREET		word processing sec. work	
7	1905 DOCK STREET		word processing	
8	1304 CRESCENT AVE		word processing	

UPCASE Function

The UPCASE function converts all letters in a character expression to uppercase.

Syntax, UPCASE function:

UPCASE(*argument*)

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

In this example, the function is placed in an assignment statement in a DATA step. You can change the values of the variable Job in place.

```
data work.upcasejob;
  set cert.temp;
  Job=upcase(job);
run;
proc print data=work.upcasejob;
run;
```

The new data set contains the converted values of Job.

Output 14.15 *Work.UpcaseJob (partial output)*

Obs	Address	. . . more variables . . .	Job	. . . more variables . . .
1	65 ELM DR		WORD PROCESSING	
2	11 SUN DR		FILING ADMIN.DUTIES	
3	712 HARDWICK STREET		ORGANIZATIONAL DEV. SPECIALIS	
4	5372 WHITEBUD ROAD		BOOKKEEPING WORD PROCESSING	
5	11 TALYN COURT		WORD PROCESSING SEC. WORK	
6	101 HYNERIAN DR		BOOKKEEPING WORD PROCESSING	
7	11 RYGEL ROAD		WORD PROCESSING	
8	121 E. MOYA STREET		WORD PROCESSING SEC. WORK	
9	1905 DOCK STREET		WORD PROCESSING	
10	1304 CRESCENT AVE		WORD PROCESSING	

LOWCASE Function

The LOWCASE function converts all letters in a character expression to lowercase.

Syntax, LOWCASE function:

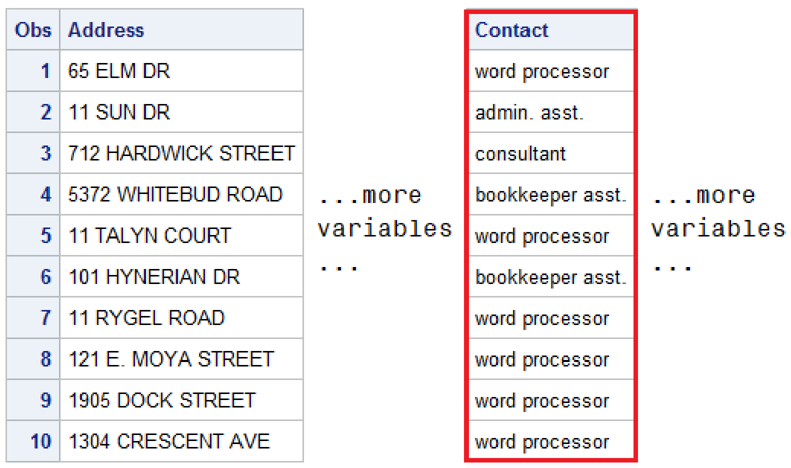
LOWCASE(*argument***)**

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

In this example, the function converts the values of the variable Contact to lowercase letters.

```
data work.lowcasecontact;
  set cert.temp;
  Contact=lowcase(contact);
run;
proc print data=work.lowcasecontact;
run;
```

Output 14.16 *Work.LowcaseContact (partial output)*



Obs	Address	...more variables...	Contact	...more variables...
1	65 ELM DR		word processor	
2	11 SUN DR		admin. asst.	
3	712 HARDWICK STREET		consultant	
4	5372 WHITEBUD ROAD		bookkeeper asst.	
5	11 TALYN COURT		word processor	
6	101 HYNERIAN DR		bookkeeper asst.	
7	11 RYGEL ROAD		word processor	
8	121 E. MOYA STREET		word processor	
9	1905 DOCK STREET		word processor	
10	1304 CRESCENT AVE		word processor	

PROPCASE Function

The PROPCASE function converts all words in an argument to proper case (so that the first letter in each word is capitalized).

Syntax, PROPCASE function:

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

- *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.

Note: If you specify *delimiter(s)*, then the default delimiters are no longer in effect.

- The PROPCASE function first converts all letters to lowercase letters and then converts the first character of words to uppercase.

- The first character of a word is the first letter of a string or any letter preceded by a default list of delimiters.

  Default delimiter List: blank / — (. tab

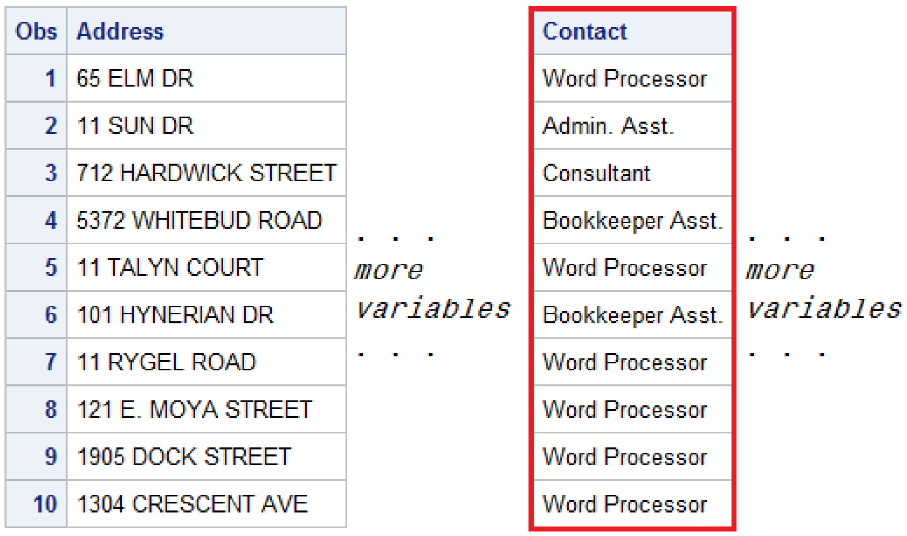
TIP Delimiters can be specified as a second argument, instead of using the default list.

In this example, the function converts the values of the variable named Contact to proper case and uses the default delimiters.

```
data work.propcasecontact;
  set cert.temp;
  Contact=propcase(contact);
run;
proc print data=work.propcasecontact;
run;
```

After the DATA step executes, the new data set is created.

Output 14.17 *Work.PropcaseContact (partial output)*


. . .
more
variables
. . .
. . .
more
variables
. . .

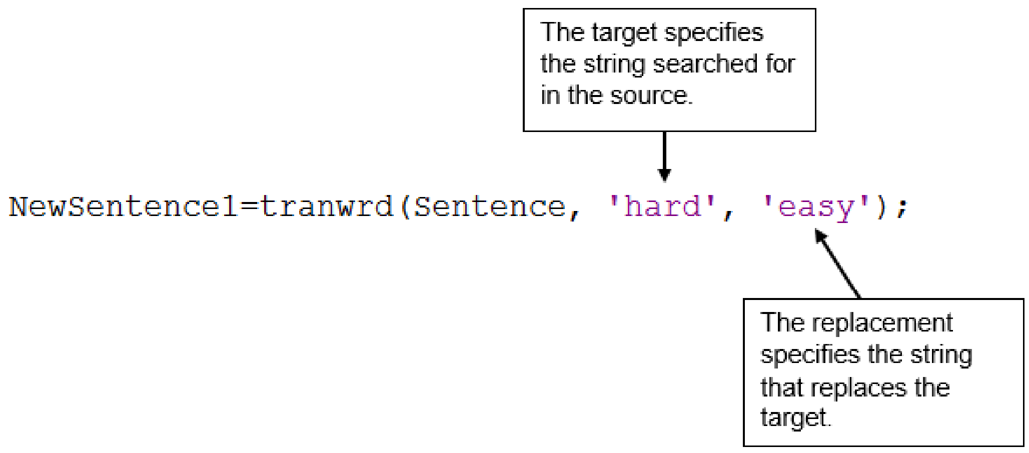
Obs	Address	Contact
1	65 ELM DR	Word Processor
2	11 SUN DR	Admin. Asst.
3	712 HARDWICK STREET	Consultant
4	5372 WHITEBUD ROAD	Bookkeeper Asst.
5	11 TALYN COURT	Word Processor
6	101 HYNERIAN DR	Bookkeeper Asst.
7	11 RYGEL ROAD	Word Processor
8	121 E. MOYA STREET	Word Processor
9	1905 DOCK STREET	Word Processor
10	1304 CRESCENT AVE	Word Processor

TRANWRD Function

Overview of the TRANWRD Function

The TRANWRD function replaces or removes all occurrences of a word in a character string. The translated characters can be located anywhere in the string.

Figure 14.7 *TRANWRD Function*


The target specifies the string searched for in the source.
NewSentence1=tranwrd(Sentence, 'hard', 'easy');
The replacement specifies the string that replaces the target.

Syntax, TRANWRD function:

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

- *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*.

Note: *target* and *replacement* can be specified as variables or as character strings. If you specify character strings, be sure to enclose the strings in quotation marks (' ' or " ").

In a DATA step, if the TRANWRD function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. To save storage space, you can add a LENGTH statement to the DATA step and specify an appropriate length for the variable. SAS sets the length of a new character variable the first time it is encountered in the DATA step. Be sure to place the LENGTH statement before the assignment statements that contain the TRANWRD function.

Example: Update Variables in Place Using TRANWRD Function

You can use TRANWRD function to update variables in place. In this example, the function updates the values of Name by changing every occurrence of the string Monroe to Manson.

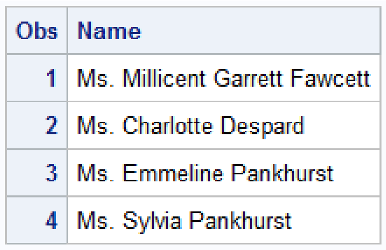
```
name=tranwrd(name,'Monroe','Manson')
```

Another example of the TRANWRD function is shown below. In this case, two assignment statements use the TRANWRD function to change all occurrences of Miss or Mrs. to Ms.

```
data work.after;
  set cert.before;
  name=tranwrd(name,'Miss','Ms.');
  name=tranwrd(name,'Mrs.','Ms.');
run;
proc print data=work.after;
run;
```

The new data set is created. The TRANWRD function changes all occurrences of Miss or Mrs. to Ms.

Figure 14.8 PROC PRINT Output of the TRANWRD Function



Obs	Name
1	Ms. Millicent Garrett Fawcett
2	Ms. Charlotte Despard
3	Ms. Emmeline Pankhurst
4	Ms. Sylvia Pankhurst

COMPBL Function

The COMPBL function removes multiple blanks from a character string by translating each occurrence of two or more consecutive blanks into a single blank.

Syntax, COMPBL function:

COMPBL(*source*)

- *source* specifies a character constant, variable, or expression to compress.

If a variable is not assigned a length before the COMPBL function returns a value to the variable, then the variable is given the length of the first argument.

The following SAS statements produce these results:

SAS Statement	Result
	`----+----1----+----2--`

`data _null_;`
`  string='Hey`
`   Diddle  Diddle';`
`  string=compbl(string);`
`  put string;`
`run;`
	`Hey Diddle Diddle`

`data _null_;`
`  string='125    E Main St';`
`  length address $10;`
`  address=compbl(string);`
`  put address;`
`run;`
	`125 E Main`

COMPRESS Function

Overview of the COMPRESS Function

The COMPRESS function returns a character string with specified characters removed from the original string. Null arguments are allowed and treated as a string with a length of zero.

The COMPRESS function returns a character string with specified characters removed from the original string.

Syntax, COMPRESS function:

COMPRESS(*source*<, *characters*> <, *modifier(s)*>**)**

- *source* specifies a character constant, variable, or expression from which specified characters are removed.
- *characters* specifies a character constant, variable, or expression that initializes a list of characters.

  By default, the characters in this list are removed from the source argument. If you specify the K modifier in the third argument, then only the characters in this list are kept in the result.

  Note: You can add more characters to this list by using other modifiers in the third argument. Enclose a literal string of characters in quotation marks.

- *modifier* specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COMPRESS function. Blanks are ignored.

  a or A
  Adds alphabetic characters to the list of characters.

  c or C
  Adds control characters to the list of characters.

  d or D
  Adds digits to the list of characters.

  f or F
  Adds the underscore character and English letters to the list of characters.

  g or G
  Adds graphic characters to the list of characters.

  h or H
  Adds a horizontal tab to the list of characters.

  i or I
  Ignores the case of the characters to be kept or removed.

  k or K
  Keeps the characters in the list instead of removing them.

  l or L
  Adds lowercase letters to the list of characters.

  n or N
  Adds digits, the underscore character, and English letters to the list of characters.

  o or O
  Processes the second and third arguments once rather than every time the COMPRESS function is called. You can use the O modifier to make the COMPRESS function more efficient when you call it in a loop, where the second and third arguments do not change.

  p or P
  Adds punctuation marks to the list of characters

  s or S
  Adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed) to the list of characters.

  t or T
  Trims trailing blanks from the first and second arguments.

  u or U
  Adds uppercase letters to the list of characters.

  w or W
  Adds printable characters to the list of characters.

  x or X
  Adds hexadecimal characters to the list of characters.

TIP If the *modifier* is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. *Modifier* can also be expressed as a variable or an expression.

Based on the number of arguments, the COMPRESS functions works as follows:

Number of Arguments	Result
only the first argument, *source*	All blanks have been removed from the argument. If the argument is completely blank, then the result is a string with a length of zero. If you assign the result to a character variable with a fixed length, then the value of that variable is padded with blanks to fill its defined length.
the first two arguments, *source* and *chars*	All characters that appear in the second argument are removed from the result.
three arguments, *source*, *chars*, and *modifier(s)*	The K modifier (specified in the third argument) determines whether the characters in the second argument are kept or removed from the result.

The COMPRESS function compiles a list of characters to keep or remove, comprising the characters in the second argument plus any types of characters that are specified by the modifiers. For example, the D modifier specifies digits. Both of the following function calls remove digits from the result:

```
compress(source, "1234567890");
compress(source, , "d");
```

To remove digits and plus or minus signs, you can use either of the following function calls:

```
compress(source, "1234567890+-");
compress(source, "+-", "d");
```

Example: Compress a Character String

```
data _null_;
  a='A B C D';
  b=compress(a);
  put b=;
run;
```

Log 14.5 *SAS Log*

```
b=ABCD
```

Example: Compress a Character String Using a Modifier

The following example uses the I modifier to ignore the case of the characters to remove.

```
data _null_;
  x='919-000-000 nc  610-000-000 pa     719-000-000 CO  419-000-000 Oh';
  y=compress(x, 'ACHONP', 'i');
put y=;
run;
```

The following is printed to the SAS log.

Log 14.6 *SAS Log*

```
y=919-000-000  610-000-000      719-000-000  419-000-000
```

Modifying Numeric Values with Functions

SAS provides additional functions to create or modify numeric values. These include arithmetic, financial, and probability functions. This book covers the following selected functions.

CEIL and FLOOR Functions

To return integers that are greater than or equal to the argument, use these 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.

Syntax, CEIL and FLOOR function:

CEIL(*argument***)**

FLOOR(*argument***)**

argument is a numeric variable, constant, or expression.

If the argument is within 1E-12 of an integer, the function returns that integer.

The following SAS statements produce this result:

Table 14.12 *CEIL and FLOOR Functions*

SAS Statement	Result
CEIL Function Examples	

`data _null_;`
`  var1=2.1;`
`  var2=-2.1;`
`  a=ceil(var1);`
`  b=ceil(var2);`
`  put "a=" a;`
`  put "b=" b;`
`run;`
	`a=3`
	
	`b=-2`

SAS Statement	Result

`data _null_;`
`  c=ceil(1+1.e-11);`
`  d=ceil(-1+1e-11);`
`  e=ceil(1+1.e-13)`
`  put "c=" c;`
`  put "d=" d;`
`  put "e=" e;`
`run;`
	`c=2`
	`d=0`
	`e=1`

`data _null_;`
`  f=ceil(223.456);`
`  g=ceil(763);`
`  h=ceil(-223.456);`
`  put "f=" f;`
`  put "g=" g;`
`  put "h=" h;`
`run;`
	`f=224`
	`g=763`
	`h=-223`

FLOOR Function Examples	

`data _null_;`
`  var1=2.1;`
`  var2=-2.1;`
`  a=floor(var1);`
`  b=floor(var2);`
`  put "a=" a;`
`  put "b=" b;`
`run;`
	`a=2`
	`b=-3`

`data _null_;`
`  c=floor(1+1.e-11);`
`  d=floor(-1+1e-11);`
`  e=floor(1+1.e-13)`
`  put "c=" c;`
`  put "d=" d;`
`  put "e=" e;`
`run;`
	`c=1`
	`d=-1`
	`e=1`

`data _null_;`
`  f=floor(223.456);`
`  g=floor(763);`
`  h=floor(-223.456);`
`  put "f=" f;`
`  put "g=" g;`
`  put "h=" h;`
`run;`
	`f=223`
	`g=763`
	`h=-224`

INT Function

To return the integer portion of a numeric value, use the INT function. Any decimal portion of the INT function argument is discarded.

Syntax, INT function:

INT(*argument*)

argument is a numeric variable, constant, or expression.

The two data sets shown below give before-and-after views of values that are truncated by the INT function.

```
data work.creditx;
  set cert.credit;
  Transaction=int(transaction);
run;
proc print data=work.creditx;
run;
```

Output 14.18 *INT Function Comparison*

```
Data Set Cert.Credit
(Before INT Function)
```

Obs	Account	Name	Type	Transaction
1	1118	ART CONTUCK	D	57.69
2	2287	MICHAEL WINSTONE	D	145.89
3	6201	MARY WATERS	C	45.00
4	7821	MICHELLE STANTON	A	304.45
5	6621	WALTER LUND	C	234.76
6	1086	KATHERINE MORRY	A	64.98
7	0556	LEE McDONALD	D	70.82
8	7821	ELIZABETH WESTIN	C	188.23
9	0265	JEFFREY DONALDSON	C	78.90
10	1010	MARTIN LYNN	D	150.55

```
Data Set Work.CreditX
(After INT Function)
```

Obs	Account	Name	Type	Transaction
1	1118	ART CONTUCK	D	57
2	2287	MICHAEL WINSTONE	D	145
3	6201	MARY WATERS	C	45
4	7821	MICHELLE STANTON	A	304
5	6621	WALTER LUND	C	234
6	1086	KATHERINE MORRY	A	64
7	0556	LEE McDONALD	D	70
8	7821	ELIZABETH WESTIN	C	188
9	0265	JEFFREY DONALDSON	C	78
10	1010	MARTIN LYNN	D	150

ROUND Function

To round values to the nearest specified unit, use the ROUND function.

Syntax, ROUND function:

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

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

If a rounding unit is not provided, a default value of 1 is used, and the argument is rounded to the nearest integer. The two data sets shown below give before-and-after views of values that are modified by the ROUND function. The first ROUND function rounds the variable AccountBalance to the nearest integer. The second ROUND function rounds the variable InvoicedAmount to the nearest tenth decimal place. The third ROUND function rounds the variable AmountRemaining to the nearest hundredth decimal place.

```
data work.rounders;
  set cert.rounders;
  AccountBalance=round(AccountBalance, 1);
  InvoicedAmount=round(InvoicedAmount, 0.1);
  AmountRemaining=round(AmountRemaining, 0.02);
  format AccountBalance InvoicedAmount PaymentReceived AmountRemaining dollar9.2;
run;
proc print data=work.rounders;
run;
```

Output 14.19 *Before and After ROUND Function*

Data Set Cert.Rounders, before the ROUND function

Obs	Account	AccountBalance	InvoicedAmount	PaymentReceived	AmountRemaining
1	1118	6246.34	967.84	1214.18	2214.18
2	2287	3687.14	607.30	4294.44	0.00
3	6201	1607.93	137.41	700.00	1045.34
4	7821	7391.62	1069.37	5000.00	3460.99
5	6621	7017.50	9334.08	8351.58	8000.00
6	1086	556.36	1537.28	1300.28	793.36
7	2556	6388.10	3577.82	6900.82	3065.10
8	7821	10872.96	3885.08	10872.96	3885.08
9	5265	1057.46	637.42	1200.00	494.88
10	1010	6387.13	0.00	3193.57	3193.56

Data Set Work.Rounders, after the ROUND function

Obs	Account	AccountBalance	InvoicedAmount	PaymentReceived	AmountRemaining
1	1118	$6,246.00	$967.80	$1,214.18	$2,214.18
2	2287	$3,687.00	$607.30	$4,294.44	$0.00
3	6201	$1,608.00	$137.40	$700.00	$1,045.34
4	7821	$7,392.00	$1,069.40	$5,000.00	$3,461.00
5	6621	$7,018.00	$9,334.10	$8,351.58	$8,000.00
6	1086	$556.00	$1,537.30	$1,300.28	$793.36
7	2556	$6,388.00	$3,577.80	$6,900.82	$3,065.10
8	7821	$10873.00	$3,885.10	$10872.96	$3,885.08
9	5265	$1,057.00	$637.40	$1,200.00	$494.88
10	1010	$6,387.00	$0.00	$3,193.57	$3,193.56

Nesting SAS Functions

To write more efficient programs you can nest functions as appropriate. You can nest any functions as long as the function that is used as the argument meets the requirements for the argument. For example, you can nest the SCAN function within the SUBSTR function in an assignment statement to compute the value for MiddleInitial:

```
MiddleInitial=substr(scan(name,3),1,1);
```

This example of nested numeric functions determines the number of years between June 15, 2018, and today:

```
Years=intck('year','15jun2018'd,today());
```