

# Chapter 13

## Transforming DATA with SAS Functions

# Overview

SAS functions are built-in routines that enable you to complete many types of data manipulations quickly and easily. Generally speaking, functions provide programming shortcuts and can be used in a DATA step anywhere that you would use a SAS expression. There are many types of SAS functions: arithmetic functions, financial functions, character functions, probability functions, and many more.

Using SAS functions, you can

- 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

This chapter concentrates on functions that

- convert data
- manipulate SAS date values
- modify values of character variables

# Some Examples of SAS Functions

SAS Functions That Compute Sample Statistics

Function	Syntax	Calculates...
SUM	<code>sum(argument, argument,...)</code>	sum of values
MEAN	<code>mean(argument, argument,...)</code>	average of nonmissing values
MIN	<code>min(argument, argument,...)</code>	minimum value
MAX	<code>max(argument, argument,...)</code>	maximum value
VAR	<code>var(argument, argument,...)</code>	variance of the values
STD	<code>std(argument, argument,...)</code>	standard deviation of the values

# General Form of SAS Functions

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

where *arguments* can be

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

Note that

- Some functions require a specific number of arguments, whereas other functions can contain any number of arguments. Some functions require no arguments.
- Even if the function does not require arguments, the function name must still be followed by parentheses (for example, *function-name()*).

# More on Function Arguments

For some functions, variable lists and arrays can also be used as arguments, as long as the list or the array is preceded by the word **OF**.

- Case of variable list:

The function

`mean(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10)`

is the same as

`mean(of x1-x10)`

- Case of array:

`mean(of newarray{*})`

# Target Variables

The MEAN function below calculates the average of three exam scores that are stored in the variables Exam1, Exam2, and Exam3. The result of a function is assigned to AvgScore, which is called a **target variable**.

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

The length of the target variable:

- Unless defined previously, a default length is assigned. The default length depends on the function; the default for character functions can be as long as **200**, which could use more space than necessary.
- 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

The variable **PayRate** in the following data set was defined as a character variable. In order to use it in computing, the data type must be converted to numeric.

SAS Data Set Hrd.Temp

City	State	Zip	Phone	StartDate	EndDate	PayRate	Days	Hours
CARY	NC	27513	6224549	14567	14621	10	11	88
CARY	NC	27513	6223251	14524	14565	8	25	200
CHAPEL HILL	NC	27514	9974749	14570	14608	40	26	208

```
data hrd.newtemp;  
  set hrd.temp (keep=city state zip phone payrate days hours);  
  Salary=payrate*hours;  
run;
```

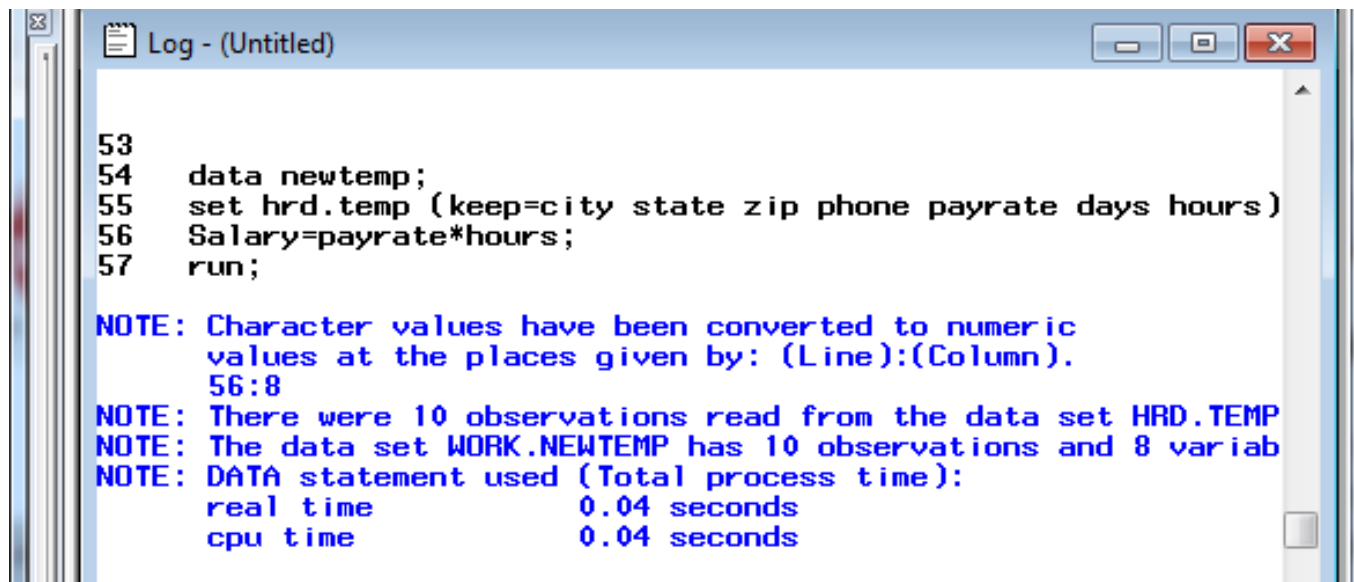
HOW?

City	State	Zip	Phone	Payrate	Days	Hours	Salary
CARY	NC	27513	6224549	10	11	88	880
CARY	NC	27513	6223251	8	25	200	1600
CHAPEL HILL	NC	27514	9974749	40	26	208	8320

# Automatic Character-to-Numeric Conversion

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 in the dataset 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 - (Untitled)

53
54   data newtemp;
55   set hrd.temp (keep=city state zip phone payrate days hours)
56   Salary=payrate*hours;
57   run;

NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      56:8
NOTE: There were 10 observations read from the data set HRD.TEMP
NOTE: The data set WORK.NEWTEMP has 10 observations and 8 variab
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```



# Automatic Character-to-Numeric Conversion

when a character value is

- assigned to a previously defined numeric variable, such as the numeric variable Rate, e.g.,  
`Rate=payrate;`
- used in an arithmetic operation, e.g.,  
`Salary=payrate*hours;`
- compared to a numeric value, using a comparison operator, e.g.,  
`if payrate>=rate;`
- specified in a function that requires numeric arguments, e.g.,  
`NewRate=sum(payrate,raise);`

The automatic conversion

- uses the **w. informat**, where w is the width of the character value that is being converted.
- 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).

# Automatic Character-to-Numeric Conversion

Character Value	automatic conversion	Numeric Value
12.47	→	12.47
-8.96	→	-8.96
1.243E1	→	12.43
1,742.64	→	.

# Automatic Character-to-Numeric Conversion

## 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 a 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;
```

```
60  data work.convtest;
61  Number=4;
62  Character='4';
63  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

64  proc print data=work.convtest;
65  where character=4;
ERROR: WHERE clause operator requires compatible variables.
66  run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.06 seconds
      cpu time            0.00 seconds

67  proc print data=work.convtest;
68  where number='4';
ERROR: WHERE clause operator requires compatible variables.
69  run;
```

# Converting Data with Functions

- SAS will detect the mismatched variables and will try an automatic character-to-numeric or numeric-to-character conversion. In previous example, the variable PayRate was automatically converted from character to numeric.
- However, this process doesn't always work. 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.
- It is *always* best to include **INPUT** and **PUT functions** in your programs to avoid data type mismatches and circumvent automatic conversion.
  - The **INPUT function** converts character data values to numeric values.
  - The **PUT function** converts numeric data values to character values.

# Explicit Character-to-Numeric Conversion

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

General form:

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

where

- *source* indicates the character variable, constant, or expression to be converted to a numeric value
- a numeric *informat* must also be specified, e.g., **input**(payrate, 2.)

```
data hrd.newtemp;  
  set hrd.temp (keep=city state zip phone payrate days hours);  
  Salary=input(payrate,2.)*hours;  
run;
```

City	State	Zip	Phone	Payrate	Days	Hours	Salary
CARY	NC	27513	6224549	10	11	88	880
CARY	NC	27513	6223251	8	25	200	1600
CHAPEL HILL	NC	27514	9974749	40	26	208	8320

# Automatic Numeric-to-Character Conversion

Numeric data values are automatically converted to character values whenever they are used in a character context. It can occur when you

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

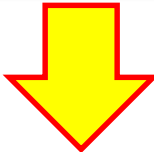
SAS writes the numeric value with the **BEST12.** format, and the resulting character value is right-aligned. This conversion occurs before the value is assigned or used with any operator or function. This can cause unexpected results. For example, suppose the original numeric value has fewer than 12 digits. The resulting character value will have 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.

# Automatic Numeric-to-Character Conversion

A new character variable named Assignment is created by concatenating the values of the numeric variable Site and the character variable Dept.

```
data hrd.newtemp;  
  set hrd.temp(keep=job  
    contact dept site);  
  Assignment=site || '/' || dept;  
run;
```

Job	Contact	Dept	Site
word processing	WORD PROCESSOR	DP	26
Filing Admin.Duties	ADMIN. ASST.	PURH	57
Organizational Dev. Specialis	CONSULTANT	PERS	34



Automatic Conversion

Job	Contact	Dept	Site	Assignment
word processing	WORD PROCESSOR	DP	26	26/DP
Filing Admin.Duties	ADMIN. ASST.	PURH	57	57/PURH
Organizational Dev. Specialis	CONSULTANT	PERS	34	34/PERS

# Explicit Numeric-to-Character Conversion

You can use the PUT function to explicitly convert numeric values to character values. General form,

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

where

- *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

Note: 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.
- For listing output, numeric formats right-align the result and character formats left-align the result.
- If you use the PUT function to create a variable that has not been previously identified, it creates a character variable whose length is equal to the format width.



# Explicit Numeric-to-Character Conversion

Now rewrite the assignment statement in your DATA step to explicitly convert the numeric values of Site to character values.

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

Job	Contact	Dept	Site
word processing	WORD PROCESSOR	DP	26
Filing Admin. Duties	ADMIN. ASST.	PURH	57
Organizational Dev. Specialist	CONSULTANT	PERS	34



Explicit Conversion

Job	Contact	Dept	Site	Assignment
word processing	WORD PROCESSOR	DP	26	26/DP
Filing Admin.Duties	ADMIN. ASST.	PURH	57	57/PURH
Organizational Dev. Specialist	CONSULTANT	PERS	34	34/PERS

# Manipulating **SAS Date Values** with Functions

**SAS date and time values:** SAS includes a variety of functions that enable you to work with SAS date and time values.

- SAS stores a **date value** as the number of days from January 1, 1960, to a given date.

Jan. 1, 1959	Jan. 1, 1960	Jan. 1, 1961
← -365	0	366 →

- A SAS **time value** is stored as the number of seconds since midnight.

(12:00 am) midnight	12:15 pm	17:00 (or 5:00 pm)
0	44100	61200 →

- A SAS **datetime value** is stored as the number of seconds between midnight on January 1, 1960, and a given date and time.

July 4, 1776 11:30:23	Jan. 1, 1960 midnight	July 4, 1994 16:10:45
← -5790400177	0	1088957445 →

# SAS Date Values

SAS stores date values as numbers so that you can easily sort the values or perform arithmetic computations. You can use SAS date values as you use any other numeric values.

```
data test;  
set hrd.temp (keep=city phone startdate enddate);  
TotDay=enddate-startdate;  
run;
```

When you execute the program, TotDay has values of 54, 41 and 38 days based on the StartDate and EndDate values in the Hrd.Temp data set.

Obs	City	Phone	Startdate	Enddate	TotDay
1	CARY	6224549	14567	14621	54
2	CARY	6223251	14524	14565	41
3	CHAPEL HILL	9974749	14570	14608	38

# Display SAS Date Values

You can display SAS date values in a variety of forms by associating a SAS format with the values. The format affects *only* the display of the dates, not the date values in the data set.

```
proc print data=hrd.temp;  
  var city phone startdate enddate;  
  format startdate enddate date9.;  
run;
```

Obs	City	Phone	Startdate	Enddate
1	CARY	6224549	19NOV1999	12JAN2000
2	CARY	6223251	07OCT1999	17NOV1999
3	CHAPEL HILL	9974749	22NOV1999	30DEC1999

## Note:

- SAS date values are valid for dates that are based on the Gregorian calendar from A.D. 1582 through A.D. 20,000.
- Use caution when working with historical dates. The Gregorian calendar was used throughout most of Europe from 1582, but Great Britain and the American colonies did not adopt the calendar until 1752.

# SAS Date Functions

## Typical Use of SAS Date Functions

Function	Typical Use	Result
<b>MDY</b>	<code>date=mdy(mon,day,yr);</code>	SAS date
<b>TODAY, DATE</b>	<code>now=today(); now=date();</code>	today's date as a SAS date
<b>TIME</b>	<code>curtime=time();</code>	current time as a SAS time

## Selected Functions to Use with SAS Date Values

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

# 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 work the same way, so we discuss them as a group.

General form:

**YEAR**(*date*)

**QTR**(*date*)

**MONTH**(*date*)

**DAY**(*date*)

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

- The **YEAR function** returns a four-digit numeric value that represents the year.
- The **QTR function** returns a value of **1, 2, 3, or 4** from a SAS date value to indicate the quarter of the year.
- The **MONTH function** returns a numeric value that ranges from **1 to 12**, representing the month of the year.
- The **DAY function** returns a numeric value from **1 to 31**, representing the day of the month.

# Finding YEAR and Month

Create a subset of the data set Hrd.Temp that contains information about all temporary employees who were hired in November 1999.

```
data hrd.nov99;  
    set hrd.temp ;  
    if year(startdate)=1999 and month(startdate)=11;  
proc print data= hrd.nov99;  
    format startdate enddate date9.;  
run;
```

Obs	City	Phone	Startdate	Enddate
1	CARY	6224549	19NOV1999	12JAN2000
2	CHAPEL HILL	9974749	22NOV1999	30DEC1999
3	DURHAM	3633618	02NOV1999	13NOV1999
4	CARRBORO	9976732	16NOV1999	04JAN2000

# WEEKDAY Function

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

```
libname library 'E:\STSCI5010\data';
proc format library=library;
  value wkfmt
    3="Tuesday"
    5="Thursday";
run;

data tuesday_thursday;
  set hrd.temp (keep=city phone
                 startdate);
  Hire_weekday=weekday(startdate);
  if weekday(startdate) in (3,5);
proc print data=tuesday_thursday;
  format startdate date9.
         Hire_weekday wkfmt.;
run;
```

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

Find out those who were hired on a Tuesday or a Thursday

City	Phone	Startdate	Hire_weekday
CARY	6223251	07OCT1999	Thursday
DURHAM	3633618	02NOV1999	Tuesday
CARRBORO	9976732	16NOV1999	Tuesday
DURHAM	3633020	06OCT1998	Tuesday
CARY	6565303	07OCT1999	Thursday
RALEIGH	4341557	29JUN1999	Tuesday



# MDY Function

The **MDY function** creates a SAS date value from numeric values that represent the month, day, and year.

General form:

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

where

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

City	State	Zip	Phone	Month	Day	Year	PayRate
CARY	NC	27513	6224549	1	12	2000	10
CARY	NC	27513	6223251	11	17	1999	8
CHAPEL HILL	NC	27514	9974749	12	30	1999	40
RALEIGH	NC	27612	6970450	10	10	1999	15

This makes it difficult to perform calculations that are based on the length of employment.

# MDY Function

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

City	State	Zip	Phone	PayRate	Days	Hours	Date
CARY	NC	27513	6224549	10	11	88	14621
CARY	NC	27513	6223251	8	25	200	14565
CHAPEL HILL	NC	27514	9974749	40	26	208	14608
RALEIGH	NC	27612	6970450	15	10	80	14527

# MDY Function

```
data dec.review2010;
  set dec.review;
  ReviewDate=mdy(12,day,2010);
run;
```

SAS Data Set Dec.Review

Site	Day	Rate	Name
Westin	12	A2	Mitchell, K
Stockton	4	A5	Worton, M
Center City	17	B1	Smith, A

SAS Data Set Dec.Review2010

Site	Day	Rate	Name	ReviewDate
Westin	12	A2	Mitchell, K	<b>18608</b>
Stockton	4	A5	Worton, M	<b>18600</b>
Center City	17	B1	Smith, A	<b>18613</b>

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

```
data dec.review2010;
  set dec.review;
  ReviewDate=mdy(15,day,2010);
run;
```

SAS Data Set Dec.Review2010

Site	Day	Rate	Name	ReviewDate
Westin	12	A2	Mitchell, K	.
Stockton	4	A5	Worton, M	.
Center City	17	B1	Smith, A	.

# DATE and TODAY Functions

The **DATE** and **TODAY** functions return the current date from the system clock as a SAS date value. The DATE and TODAY functions have the same form and can be used interchangeably. These functions require no arguments.

General form:           **DATE()**  
                              **TODAY()**

Enddate	EditDate
14621	19651
14565	19651
14608	19651
14527	19651
14561	19651
14613	19651
14108	19651
14162	19651
14172	19651
14182	19651

*/\*Use 10/20/2013 as today's date\*/*

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

# INTCK Function

The **INTCK function** returns the number of time intervals that occur in a given time span. You can use it to count the passage of days, weeks, months, and so on.

General form: **INTCK('interval', from, to)**

where

- **'interval'** specifies a character constant or variable. The value can be one of the following: DAY, WEEKDAY, WEEK, TENDAY, SEMIMONTH, MONTH, QTR, SEMIYEAR, and 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.

# INTCK Function

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.

- WEEK intervals are counted by Sundays rather than seven-day multiples from the *from* argument.
- MONTH intervals are counted by day 1 of each month.
- YEAR intervals are counted from 01JAN, not in 365-day multiples.

SAS Statement	Value
Weeks=intck('week','31dec2000'd,'01jan2001'd);	
Months=intck('month','31dec2000'd,'01jan2001'd);	
Years=intck('year','31dec2000'd,'01jan2001'd);	
Years=intck('year','15jun1999'd,'15jun2001'd);	
Months=intck('month','15jun1999'd,'15jun2001'd);	
Years=intck('year','01jan2002'd,'31dec2002'd);	

# INTCK Function: More Example

A common use of the INTCK function is to identify periodic events such as anniversaries and due dates . The following program identifies mechanics whose 35th 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.anniv35;  
    set flights.mechanics(keep=id lastname firstname hired);  
    Years=intck('year', hired, today());  
    if years=35 and month(hired)=month(today());  
run;  
proc print data=work.anniv35;  
    title '35-Year Anniversaries This Month';  
run;
```

## 35-Year Anniversaries This Month

Obs	ID	LastName	FirstName	Hired	Years
1	1400	APPLE	TROY	19OCT78	35

# INTNX 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.

General form: **INTNX**('interval', start-from, increment<,'alignment'>)  
where

- *'interval'* specifies a character constant or variable. The value can be one of the following: DAY, WEEKDAY, WEEK, TENDAY, SEMIMONTH, MONTH, QTR, SEMIYEAR, YEAR, DATETIME and TIME.
- *Start-from* specifies a SAS date, time, or datetime value that identifies the beginning of the time span.
- *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 (default), middle, or end of the interval.



# Example: INTNX Function

The following statement creates the variable TargetYear and assigns it a SAS date value of **13515**, which corresponds to January 1, 1997.

```
TargetYear=intnx('year', '05feb94'd, 3);
```

The following statement assigns the value (**15157**) for the date July 1, 2001, to the variable TargetMonth.

```
TargetMonth=intnx('semiyear', '01jan2001'd, 1);
```

# Using an **Alignment** in INTNX Function

The purpose of the optional **alignment** value: it lets you specify whether the returned value should be at the beginning, middle, end or the same day 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**

SAS Statement	Date Value	
MonthX=intnx('month','01jan1995'd,5,'b');	12935	(June 1, 1995)
MonthX=intnx('month','01jan1995'd,5,'m');	12949	(June 15, 1995)
MonthX=intnx('month','01jan1995'd,5,'e');	12964	(June 30, 1995)
MonthX=intnx('month','10jan1995'd,5, 's');	12944	(June 10, 1995)

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.

General form:     **DATDIF**(*start\_date*, *end\_date*, *basis*)  
                      **YRDIF**(*start\_date*, *end\_date*, *basis*)

where

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

# DATDIF and YRDIF Functions: Valid Character String of Basis

There are **two** valid character strings for basis in the DATDIF function and **four** valid character strings for basis in the YRDIF 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 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

YRDIF Statement	Returned Value
yrdif('16oct1998'd, '16feb2003'd, '30/360')	4.333333333
yrdif('16oct1998'd, '16feb2003'd, 'ACT/ACT')	4.3369863014
yrdif('16oct1998'd, '16feb2003'd, 'ACT/360')	4.4
yrdif('16oct1998'd, '16feb2003'd, 'ACT/365')	4.3397260274

# Modifying Character Values with Functions

Character values can be manipulated with SAS functions, including:

- replace the contents of a character value
- trim trailing blanks from a character value
- search a character value and extract a portion of the value
- convert a character value to uppercase or lowercase

Examples:

SAS Data

<b>Name</b>	<b>LastName</b>	<b>FirstName</b>	<b>MiddleName</b>
CICHOCK, ELIZABETH MARIE ▶	CICHOCK	ELIZABETH	MARIE

<b>Phone</b>		<b>Phone</b>
6224549		<b>433</b> 4549

# Selected Character Functions

Function	Purpose
SCAN	returns a specified word from a character value.
SUBSTR	extracts a substring or replaces character values.
TRIM	trims trailing blanks from character values.
CATX	concatenates character strings, removes leading and trailing blanks, and inserts separators.
INDEX	searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string.
FIND	searches for a specific substring of characters within a character string.
UPPERCASE	converts all letters in a value to uppercase.
LOWCASE	converts all letters in a value to lowercase.
PROPCASE	converts all letters in a value to proper case.
TRANWRD	replaces or removes all occurrences of a pattern of characters within a character string.

# The SCAN Function

The SCAN function enables you to separate a character value into words and to return a specified word. It uses **delimiters**, which are characters that are specified as word separators, to separate a character string into words. Then the function returns whichever word you specify.

General form:

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

where

- *argument* specifies the character variable or expression to scan.
- *n* specifies which word to return, e.g., if *n* = 1, it returns the first word.
- *delimiters* are special characters that must be enclosed in quotation marks (' ' or " "). If you do not specify *delimiters*, default delimiters are used, which are **. < ( + | & ! \$ \* ) ; ^ - / , % , and blank**. When you specify multiple delimiters, SAS uses any of the delimiters, *singly or in any combination*, as word separators. The SCAN function treats two or more contiguous delimiters as one delimiter.

# The SCAN Function: Example

SAS Data Set Hrd.Temp

Agency	ID	Name
Administrative Support, Inc.	F274	CICHOCK, ELIZABETH MARIE
Administrative Support, Inc.	F101	BENINCASA, HANNAH LEE
OD Consulting, Inc.	F054	SHERE, BRIAN THOMAS

```
data hrd.newtemp(drop=name);  
  set hrd.temp;  
  LastName=scan(name,1);  
  FirstName=scan(name,2);  
  MiddleName=scan(name,3);  
run;
```

SAS Data Set Hrd.Newtemp

Agency	ID	LastName	FirstName	MiddleName
Administrative Support, Inc.	F274	CICHOCK	ELIZABETH	MARIE
Administrative Support, Inc.	F101	BENINCASA	HANNAH	LEE
OD Consulting, Inc.	F054	SHERE	BRIAN	THOMAS



# The SCAN Function: Variable Length

- By default, in SAS 9.4, the length of the target variable that has not been assigned a length is given a length of the SCAN function's argument, here name, which is longer than necessary .
- To save storage space, add a LENGTH statement to your DATA step, and specify an appropriate length for all the variables. Be sure to place the LENGTH statement *before* the assignment statements that contain the SCAN function.

```
data hrd.newtemp(drop=name);  
    set hrd.temp;  
    length LastName FirstName MiddleName $ 10;  
    LastName=scan(name,1);  
    FirstName=scan(name,2);  
    MiddleName=scan(name,3);  
run;
```

# The SUBSTR Function

The SUBSTR function can be used to extract a portion of a character value or to replace the contents of a character value.

General form:

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

where

- *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.

# The SUBSTR Function: Example

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

SAS Data Set Hrd.Newtemp

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

```
data work.newtemp(drop=middlename);  
  set hrd.newtemp;  
  length MiddleInitial $ 1;  
  MiddleInitial=substr(middlename,1,1);  
run;
```

SAS Data Set Work.Newtemp

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

# Replacing Text Using SUBSTR

This function can also be used to replace the contents of a character variable. The position of the function is on the *left side* of an assignment statement.

SAS Data Set Hrd.Temp

City	State	Zip	Phone	StartDate	EndDate	PayRate	Days	Hours
CARY	NC	27513	6224549	14567	14621	10	11	88
CARY	NC	27513	6223251	14524	14565	8	25	200
CHAPEL HILL	NC	27514	9974749	14570	14608	40	26	208

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

SAS Data Set Hrd.Temp2

City	State	Zip	Phone	StartDate	EndDate	PayRate	Days	Hours
CARY	NC	27513	4334549	14567	14621	10	11	88
CARY	NC	27513	4333251	14524	14565	8	25	200
CHAPEL HILL	NC	27514	9974749	14570	14608	40	26	208

# Replacing Text Using SUBSTR: Another Example

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

Test		Test
S73 <b>81</b> K2	→	S73 <b>92</b> K2
S73 <b>81</b> K7	→	S73 <b>92</b> K7

# SCAN versus SUBSTR

- ❖ The **SUBSTR** function is similar to the **SCAN** function. 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.
- ❖ The **SCAN** function is best used when 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(s).

# The TRIM Function

The TRIM function removes trailing blanks from character values.

General form:

**TRIM**(*argument*)

Where

- *argument* can be any character expression, such as
  - a character variable
  - another character function

Note: When the result of the Trim() function is assigned to a target variable, which has not been assigned a length, the length of the target variable is the same as the length of argument of the Trim() function.

# Without Using the TRIM Function

```
data hrd.newtemp(drop=address city state zip);  
  set hrd.temp;  
  NewAddress=address||', '||city||', '||zip;  
run;
```

SAS Data Set Hrd.NewTemp

NewAddress			
65 ELM DRIVE	, CARY	, 27513	
11 SUN DRIVE	, CARY	, 27513	
712 HARDWICK STREET	, CHAPEL HILL	, 27514	
5372 WHITEBUD ROAD	, RALEIGH	, 27612	

The values of the new variable contain embedded blanks because the values of the original address variables contained trailing blanks. Whenever the value of a character variable is shorter than the length of the variable, SAS pads the value with trailing blanks.



# Using the TRIM Function

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

SAS Data Set Hrd.Newtemp

NewAddress
65 ELM DRIVE, CARY, 27513
11 SUN DRIVE, CARY, 27513
712 HARDWICK STREET, CHAPEL HILL, 27514
5372 WHITEBUD ROAD, RALEIGH, 27612

The trailing blanks were removed.

# The TRIM Function: Points to Remember

The TRIM function does not affect how a variable is stored. Suppose you trim the values of a variable and then assign these values to a new variable. The trimmed values are padded with trailing blanks again if the values are shorter than the length of the new variable.

Here's an example. In the DATA step below, the trimmed value of Address is assigned to the new variable Street. When the trimmed value is assigned to Street, trailing blanks are added to the value to match the length of 20.

```
data temp;  
  set hrd.temp;  
  length Street $ 20;  
  Street=trim(address);  
run;
```

Address length=32	Street length=20
65 ELM DRIVE.....	65 ELM DRIVE.....
11 SUN DRIVE.....	11 SUN DRIVE.....
712 HARTWICK STREET.....	712 HARTWICK STREET.

# The CATX Function

- Beginning in SAS®9, the **CATX** function concatenates character strings, remove leading and trailing blanks, and insert separators. It returns a value to a variable or 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.
- If the CATX 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 your DATA step, and specify an appropriate length for your variable.
- If the concatenation operator (||) returns a value to a variable that has not previously been assigned a length, then that variable is given a length that is the sum of the lengths of the values which are being concatenated.

# The CATX Function

General form:

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

where

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

```
data hrd.newtemp(drop=address city state zip);  
  set hrd.temp;  
  NewAddress=catx(' ', address, city, zip);  
run;
```

SAS Data Set Hrd.Newtemp

NewAddress
65 ELM DRIVE, CARY, 27513
11 SUN DRIVE, CARY, 27513
712 HARDWICK STREET, CHAPEL HILL, 27514
5372 WHITEBUD ROAD, RALEIGH, 27612

The same result as  
earlier (Slide 49)

# The INDEX Function

The INDEX function searches a character value for a specified string 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.

General form:

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

where

- *source* specifies the character variable or expression to search
- *excerpt* specifies the target character string that is enclosed in quotation marks (' ' or " "), so it is case sensitive (but you can use UPCASE or LOWCASE functions to uniform these cases)

# The INDEX Function: Example

Shows the temporary employees who have word processing experience.

```
data hrd.datapool;  
  set hrd.temp;  
  if index(job, 'word processing') > 0;  
run;
```

SAS Data Set Hrd.Datapool

Job	Contact	Dept	Site
word processing	WORD PROCESSOR	DP	26
bookkeeping, word processing	BOOKKEEPER AST	BK	57
word processing, sec. work	WORD PROCESSOR	DP	95
bookkeeping, word processing	BOOKKEEPER AST	BK	44
word processing	WORD PROCESSOR	DP	59
word processing, sec. work	WORD PROCESSOR	PUB	38
word processing	WORD PROCESSOR	DP	44
word processing	WORD PROCESSOR	DP	90

# The FIND Function

Beginning in SAS®9, the **FIND** function searches for a specific substring of characters within a character string for the first occurrence of the substring, and returns the position of that substring. If the substring is not found in the string, FIND returns a value of 0, which is similar to INDEX.

General form:

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

where

- *string* specifies a character constant, variable, or expression that will be searched for substrings
- *substring* is a character constant, variable, or expression that specifies the substring of characters to search for in *string*
- *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 (**positive**, to the right; **negative**, to the left). The default value for *startpos* is 1.

# The FIND Function: More About Modifiers

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.
- 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.



# The FIND Function: Examples

SAS Statements	Results
whereisshe=find('She sells seashells? Yes, she does.', 'she ');	27
variable1='She sells seashells? Yes, she does.'; variable2='she '; variable3='i'; whereisshe_i=find(variable1,variable2,variable3);	1
expression1='She sells seashells? '    'Yes, she does.'; expression2=scan('he or she',3)    ' '; expression3=trim('t '); whereisshe_t=find(expression1,expression2,expression3);	14
xyz='She sells seashells? Yes, she does.'; startposvar=22; whereisshe_22=find(xyz,'she', startposvar);	27
xyz='She sells seashells? Yes, she does.'; startposexp=1-23; whereisShe_ineg22=find(xyz,'She','i',startposexp);	14

# The LOWCASE and UPCASE Functions

```
data hrd.newtemp;  
  set hrd.temp;  
  Job=upcase(job);  
run;
```

SAS Data Set Hrd.Newtemp

Job	Contact	Dept	Site
<b>WORD PROCESSING</b>	WORD PROCESSOR	DP	26
<b>FILING, ADMINISTRATIVE DUTIES</b>	ADMIN. ASST.	PURH	57
<b>ORGANIZATIONAL DEV. SPECIALIST</b>	CONSULTANT	PERS	34
<b>BOOKKEEPING, WORD PROCESSING</b>	BOOKKEEPER ASST.	BK	57

```
data hrd.newtemp;  
  set hrd.temp;  
  Contact=lowcase(contact);  
run;
```

SAS Data Set Hrd.Newtemp

Job	Contact	Dept	Site
word processing	<b>word processor</b>	DP	26
filing, administrative duties	<b>admin. asst.</b>	PURH	57
organizational dev. specialist	<b>consultant</b>	PERS	34

# The PROPCASE Function

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

General form:

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

where

- *argument* can be any SAS expression, such as a character variable or constant
- *delimiter(s)* specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab. If you specify a *delimiter(s)*, then the default delimiters are no longer in effect.

The PROPCASE function copies a character argument and converts all uppercase letters to lowercase letters. It then converts to uppercase the first character of a word that is preceded by a delimiter. PROPCASE uses the default delimiters unless you use the *delimiter(s)* argument.

# The PROPCASE Function: Example

```
data hrd.newtemp;  
  set hrd.temp;  
  Contact=propcase(contact);  
Run;
```

SAS Data Set Hrd.Newtemp

Job	Contact	Dept	Site
word processing	<b>Word Processor</b>	DP	26
filing, administrative duties	<b>Admin. Asst.</b>	PURH	57
organizational dev. specialist	<b>Consultant</b>	PERS	34

# The TRANWRD Function

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

General form:

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

where

- *source* specifies the source string that you want to translate
- *target* specifies the string that SAS searches for in *source*
- *replacement* specifies the string that replaces *target*.
- *target* and *replacement* can be specified as variables or as character strings. If you specify character strings, be sure to enclose the strings in quotation marks (' ' or " ").

# The TRANWRD Function: Example

Use TRANWRD function to update variables in place

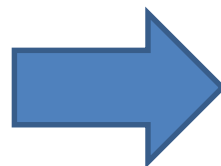
```
data work.after;  
  set work.before;  
  name=tranwrd(name,'Miss','Ms.');
```

**name=tranwrd**(name,'Mrs.','Ms.');

```
run;
```

SAS Data Set Work.Before

Name
Mrs. Millicent Garrett Fawcett
Miss Charlotte Despard
Mrs. Emmeline Pankhurst
Miss Sylvia Pankhurst



SAS Data Set Work.After

Name
Ms. Millicent Garrett Fawcett
Ms. Charlotte Despard
Ms. Emmeline Pankhurst
Ms. Sylvia Pankhurst

# Modifying Numeric Values with Functions

SAS provides many functions to create or modify numeric values. These include arithmetic, financial, and probability functions. For example,

- **INT Function**

The INT function returns the integer portion of a numeric value. Any decimal portion of the INT function argument is discarded. General form:

**INT**(*argument*)

where

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

- **ROUND Function**

The ROUND function rounds values to the nearest specified unit. General form:

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

where

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

*round-off-unit* is numeric and nonnegative.

# Modifying Numeric Values with Functions: Examples

SAS Data Set  
Work.Before

Examples
326.54
98.20
-32.66
1401.75

```
data work.after;  
  set work.before;  
  Examples=int(examples);  
run;
```

SAS Data Set  
Work.After

Examples
326
98
-32
1401

SAS Data Set  
Work.Before

Examples
326.54
98.20
-32.66
1401.75

```
data work.after;  
  set work.before;  
  Examples=round(examples,.2);  
run;
```

SAS Data Set  
Work.After

Examples
326.6
98.2
-32.6
1401.8

(Rounds the first argument to the nearest multiple of the second argument; take the greater value if the argument is right in the middle.)



# Nesting SAS Functions

To write more efficient programs you can **nest** functions, such as combining the two functions:

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

into one nested expression:

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