## SAS ${ }^{\circ}$ Functions by Example Second Edition

Ron Cody

SAS ${ }^{\circ}$ Press

## Gsas

## Contents

## List of Programs vii <br> Preface to the Second Edition xix Preface to the First Edition xxi <br> Acknowledgments xxiii Introduction xxv

## Chapter 1 Character Functions 1

Introduction 3
Functions That Change the Case of Characters 5
Functions That Remove Characters from Strings 11
Functions That Search for Characters 16
Functions That Extract Parts of Strings 46
Functions That Join Two or More Strings Together 57
Functions That Remove Blanks from Strings 70
Functions That Compare Strings (Exact and "Fuzzy" Comparisions 78
Functions That Divide Strings into "Words" 97
Functions That Substitute Letters or Words in Strings 106
Functions That Compute the Length of Strings 111
Functions That Count the Number of Letters or Substrings in a String 114
Miscellaneous String Functions 121
Chapter 2 Perl Regular Expressions 143
Introduction 143
A Brief Tutorial on Perl Regular Expressions 144
Function That Defines a Regular Expression 150
Functions That Locate Text Patterns 152
Function That Substitutes One String for Another 168
Function That Releases Memory Used by a Regular Expression 171
iv Contents
Chapter 3 Sort Functions ..... 173
Introduction ..... 173
Function That Sorts Numeric Values ..... 174
Function That Sorts Character Values ..... 176
Chapter 4 Date and Time Functions ..... 179
Introduction ..... 180
Functions That Create SAS Date, Datetime, and Time Values ..... 180
Functions That Extract the Year, Month, Day, etc., from a SAS Date ..... 188
Functions That Extract Hours, Minutes, and Seconds from SAS
Datetimes and Time Values ..... 192
Functions That Extract the Date or Time from SAS Datetime Values ..... 195
Functions That Work with Date, Datetime, and Time Intervals ..... 197
Function That Computes Dates of Standard Holidays ..... 212
Functions That Work with Julian Dates ..... 214
Chapter 5 Array Functions ..... 219
Introduction ..... 219
Chapter 6 Truncation Functions ..... 225
Introduction ..... 225
Functions That Round and Truncate Numerical Values ..... 226
Function That Returns SAS Numerical Values Stored in Fewer than 8 Bytes ..... 234
Chapter 7 Descriptive Statistics Functions ..... 237
Introduction ..... 238
Functions That Determine the Number of Missing and Non-missingValues in a List of SAS Variables238
Functions That Compute Sums, Means, and Medians ..... 241
Functions That Compute the Spread or Dispersion ofData Values244
Functions That Determine the Ordering of Data Values ..... 247
Chapter 8 Mathematical and Numeric Functions ..... 267
Introduction ..... 267
Commonly Used Mathematical Functions ..... 268
Functions That Work with Exponentiation and Logarithms ..... 276
Factorial and Gamma Functions ..... 279
Miscellaneous Functions ..... 282
Chapter 9 Random Number Functions ..... 287
Introduction ..... 287
Functions That Generate Uniform Random Numbers ..... 290
Functions That Generate Normally Distributed Numbers ..... 303
A Function That Generates Random Numbers from a Wide Variety of Probability Distributions 310
Chapter 10 Special Functions ..... 315
Introduction ..... 315
Functions That Obtain Values from Previous Observations ..... 315
Functions That Perform Character-to-Numeric or Numeric-to- Character Conversion ..... 321
Function That Sets Variable Values to a SAS Missing Value 333
Chapter 11 State and ZIP Code Functions ..... 335
Introduction ..... 335
Functions That Convert FIPS Codes ..... 336
Functions That Convert State Codes ..... 339
Functions That Convert ZIP Codes ..... 341
Functions That Compute Geographical Distance ..... 346
Chapter 12 Trigonometric Functions ..... 351
Introduction ..... 351
Three Basic Trigonometric Functions ..... 351
Three Inverse Trigonometric Functions ..... 355
Chapter 13 Macro Functions ..... 359
Introduction ..... 359
vi Contents
Chapter 14 SAS File I/O Functions ..... 371
Introduction ..... 371
Functions That Determine if SAS Data Sets Exist, and That Openand Close Files 373
Functions That Return SAS Data Set Characteristics ..... 375
Functions That Return Variable Information ..... 381
Chapter 15 Variable Information Functions ..... 391
Introduction ..... 391
Functions That Determine SAS Variable Information ..... 392
Functions That Determine Format Information ..... 403
Functions That Determine Informat Information ..... 406
Chapter 16 Bitwise Logical Functions ..... 411
Introduction ..... 411
List of Functions ..... 421
Index ..... 429

## Chapter 4

## Date and Time Functions

Introduction ..... 180
Functions That Create SAS Date, Datetime, and Time Values ..... 180
MDY 180 TODAY ..... 184
DHMS 183 DATETIME ..... 185
HMS ..... 184
DATE ..... 184
Functions That Extract the Year, Month, Day, etc., from a SAS Date ..... 188
YEAR 188 WEEK 190QTR 189 WEEKDAY191
MONTH 189 DAY 191
Functions That Extract Hours, Minutes, and Seconds from SAS Datetime and TimeValues192
HOUR ..... 193
MINUTE ..... 193
SECOND ..... 194
Functions That Extract the Date or Time from SAS Datetime Values ..... 195
DATEPART ..... 195
TIMEPART ..... 196
Functions That Work with Date, Datetime, and Time Intervals ..... 197
INTCK ..... 197
INTNX ..... 205
YRDIF ..... 209
Function That Computes Dates of Standard Holidays ..... 212
HOLIDAY ..... 212
Functions That Work with Julian Dates ..... 214
DATEJUL ..... 214
JULDATE ..... 215
JULDATE7 ..... 216

## Introduction

Before you start working with SAS date and time functions, remember that SAS date values are the number of days between January 1, 1960, and a specified date. Dates after January 1, 1960, are stored as positive numbers; dates before January 1, 1960, are stored as negative numbers. SAS time values are the number of seconds between midnight of the current day and another time value. SAS datetime values are the number of seconds between midnight, January 1,1960 , and the specified date and time. Some of the more commonly used date functions extract the day of the week, the month, or the year from a SAS date value.

Other functions deal with intervals, either the number of intervals between two dates or the date after a given number of intervals have passed. You can even compute the number of working days (the default is Saturday and Sunday as non-working days) between two dates. Making this calculation even more useful is the HOLIDAY function that, given a year, returns the date for many of the major holidays.

For situations where you only have month, day, and year values but do not have a SAS date, the MDY function can create a SAS date value, given a value for the month, day, and year. Now let's get started.

## Functions That Create SAS Date, Datetime, and Time Values

The first three functions in this group of functions create SAS date values, datetime values, and time values from the constituent parts (month, day, year, hour, minute, second). The DATE and TODAY functions are equivalent and they both return the current date. The DATETIME and TIME functions are used to create SAS datetime and time values, respectively.

## Function: MDY

Purpose: To create a SAS date from the month, day, and year.
Syntax: MDY(month, day, year)
month is a numeric variable or constant representing the month of the year (a number from 1 to 12).
day is a numeric variable or constant representing the day of the month (a number from 1 to 31 ).
year is a numeric variable or constant representing the year.
Values of month, day, and time that do not define a valid date result in a missing value, and an error message is written to the SAS log.

## Examples

For these examples, $\mathrm{M}=11, \mathrm{D}=15, \mathrm{Y}=2003$.

| Function | Returns |
| :--- | :--- |
| MDY (M, D, Y) | $16024 \quad(15 \mathrm{NOV} 2003-$ formatted value $)$ |
| MDY $(10,21,1980)$ | $7599 \quad(210 \mathrm{CT1980}-$ formatted value $)$ |
| MDY $(1,1,1950)$ | $-3652 \quad$ (01 JAN1950 - formatted value) |
| MDY $(13,01,2003)$ | numeric missing value |

## Program 4.1: Creating a SAS date value from separate variables representing the day, month, and year of the date

```
    ***Primary function: MDY;
    data funnydate;
        input @1 Month 2.
            @7 Year 4.
            @13 Day 2.;
        Date = mdy(Month,Day,Year);
        format Date mmddyy10.;
    datalines;
    05 2000 25
    11 2001 02
;
title "Listing of FUNNYDATE";
proc print data=funnydate noobs;
run;
```


## Explanation

Here the values for month, day, and year were not in a form where any of the standard date informats could be used. Therefore, the day, month, and year values were read into separate variables and the MDY function was used to create a SAS date. See the following listing:

| Listing | of | FUNNYDATE |  |
| :---: | :---: | ---: | ---: |
| Month | Year | Day | Date |
|  |  |  |  |
| 5 | 2000 | 25 | $05 / 25 / 2000$ |
| 11 | 2001 | 2 | $11 / 02 / 2001$ |

Program 4.2: Program to read in dates and set the day of the month to 15 if the day is missing from the date

```
***Primary function: MDY;
***Other functions: SCAN, INPUT, MISSING;
data missing;
        input @1 Dummy $10.;
        Day = scan(Dummy, 2,'/');
        if not missing(Day)then Date = input(Dummy,mmddyy10.);
        else Date = mdy(input(scan(Dummy,1,'/'),2.),
                        15,
                input(scan(Dummy,3,'/'),4.));
        format date date9.;
    datalines;
    10/21/1946
    1/ /2000
    01/ /2002
    ;
    title "Listing of MISSING";
    proc print data=missing noobs;
    run;
```


## Explanation

This program reads in a date and, when the day of the month is missing, it uses the 15th of the month. If the date was already stored as a character string in a SAS data set, this approach would work well.

The entire date is first read as a character string as the variable DUMMY. Next, the SCAN function is executed with the slash character (/) as the "word" delimiter. The second word is the month. If this is not missing, the INPUT function is used to convert the character string into a SAS date.

If DAY is missing, the MDY function is used to create the SAS date, with the value of 15 representing the day of the month. The listing follows:

| Listing of MISSING |  |  |
| :--- | ---: | ---: |
| Dummy | Day | Date |
|  |  |  |
| $10 / 21 / 1946$ | 21 | 210CT1946 |
| $1 / / 2000$ |  | 15JAN2000 |
| $01 / / 2002$ |  | 15JAN2002 |

## Function: DHMS

Purpose: To create a SAS datetime value from a SAS date value and a value for the hour, minute, and second.

Syntax: DHMS (date, hour, minute, second)
date is a SAS date value, either a variable or a date constant.
hour is a numerical value for the hour of the day. If hour is greater than 24, the function will return the appropriate datetime value.
minute is a numerical value for the number of minutes.
second is a numerical value for the number of seconds.
Values of the date value that are invalid result in a missing value, and an error message is written to the SAS log.

## Examples

For these examples, DATE = '02JAN1960'D, H = 23, $\mathrm{M}=15, \mathrm{~S}=30$.

| Function | Returns |
| :--- | :--- |
| DHMS (DATE, H, M, S) | 170130 (02JAN60:23:15:30 - formatted) |
| DHMS ('04JUN2003'd,25,12,12) | 1370394732 (05JUN03:01:12:12 - formatted) |
| DHMS ('01JAN1960'd,0,70,0) | 4200 (01JAN60:01:10:00 - formatted) |

See Program 4.3.

## Function: HMS

Purpose: To create a SAS time value from the hour, minute, and second.
Syntax: HMS (hour, minute, second)
hour is the value corresponding to the number of hours.
minute is the value corresponding to the number of minutes.
second is the value corresponding to the number of seconds.

## Examples

For these examples, $\mathrm{H}=1, \mathrm{M}=30$, $\mathrm{S}=15$.

| Function | Returns |
| :--- | :--- |
| HMS $(H$, M, S) | $5415(1: 30: 15$ - formatted value $)$ |
| HMS (0, 0, 23) | $23(0: 00: 23$ - formatted value $)$ |

See Program 4.3.

## Function: DATE and TODAY (equivalent functions)

Purpose: To return the current date.
Syntax: DATE () or TODAY ()
Note that the parentheses are needed even though these functions do not take any arguments. (What did the TODAY function say to the MEAN function? "Don't give me any arguments!")

## Examples

Note: This function was run on June 4, 2003.

| Function | Returns |
| :--- | :--- |
| DATE ( $)$ | $15860(04$ JUN2003 - formatted) |
| TODAY ( $)$ | 15860 (04 JUN2003 - formatted) |

See Program 4.3.

## Function: DATETIME

Purpose: To return the datetime value for the current date and time.
Syntax: DATETIME ()

## Examples

Note: This function was run at 8:10 PM on June 4, 2004.

| Function | Returns |  |
| :--- | :--- | :--- |
| DATETIME () | 1370376600 | (04JUN03:20:10:00 - formatted) |

See Program 4.3.

## Function: TIME

Purpose: To return the time of day when the program was run.
Syntax: TIME ()

## Examples

Note: This function was run at 8:10 PM.

| Function | Returns |
| :--- | :--- |
| $\operatorname{TIME}()$ | $72600 \quad(20: 10: 00-$ formatted $)$ |

## Program 4.3: Determining the date, datetime value, and time of day

```
***Primary functions: DHMS, HMS, TODAY, DATETIME, TIME, YRDIF
***Other functions: INT;
data test;
        Date = today();
        DT = datetime();
        Time = time();
        DT2 = dhms(Date, 8,15,30);
        Time2 = hms(8,15,30);
        DOB = '01jan1960'd;
        Age = int(yrdif(DOB,Date,'actual'));
        format Date DOB date9. DT DT2 datetime. Time Time2 time.;
    run;
    title "Listing of Data Set TEST";
    proc print data=test noobs;
    run;
```


## Explanation

This program was run in the morning of November 10, 2009, so the values for the date, datetime, and time values correspond to that date and time.

The variable DT2 is a SAS datetime value created from the current date and specified values for the hour, minute, and second. TIME2 is a SAS time value created from three values for hour, minute, and second.

Finally, the age was computed using the YRDIF function. (See details and an important note on the YRDIF function later in this chapter.) The INT function was used to compute age as of the last birthday (it throws away all digits to the right of the decimal point). Please see the following listing:


## Creating a Data Set to Demonstrate Other Date Functions

Run Program 4.4 to create a SAS data set called DATES. A listing of this data set follows.

## Program 4.4: Program to create the DATES data set

```
    data dates;
        informat Date1 Date2 date9.;
        input Date1 Date2;
        format Date1 Date2 date9.;
    datalines;
    01JAN1960 15JAN1960
    02MAR1961 18FEB1962
    25DEC2000 03JAN2001
    01FEB2002 31MAR2002
    ;
    title "Listing of Data Set DATES";
    proc print data=dates noobs;
    run;
```


## Explanation

Although this is not a function example program, one feature should be explained: Since the INPUT statement is reading list input (i.e., one or more spaces between the data values) and since you need to supply an informat so that the values will be read as SAS date values, an INFORMAT statement precedes the INPUT statement, indicating that both variables, DATE1 and DATE2, should be read with the DATE9. informat.

```
Listing of Data Set DATES
    Date1 Date2
01JAN1960 15JAN1960
02MAR1961 18FEB1962
25DEC2000 03JAN2001
01FEB2002 31MAR2002
```


## Functions That Extract the Year, Month, Day, etc. from a SAS Date

This group of functions takes a SAS date value and returns parts of the date, such as the year, the month, or the day of the week. Since these functions are demonstrated in a single program, let's supply the syntax and examples.

Function: YEAR
Purpose: To extract the year from a SAS date.
Syntax: YEAR(date)
date is a SAS date value.
Examples

| Function | Returns |
| :--- | :--- |
| YEAR ('16AUG2002'd) | 2002 |
| YEAR ('16AUG02'd) | 2002 |

See Program 4.5.

## Function: QTR

Purpose: $\quad$ To extract the quarter (January-March $=1$, April-June $=2$, etc.) from a SAS date.

Syntax: QTR(date)
date is a SAS date value.

## Examples

| Function | Returns |
| :--- | :--- |
| QTR ('05FEB2003'd) | 1 |
| QTR('01DEC2003'd) | 4 |

See Program 4.5.

## Function: MONTH

Purpose: To extract the month of the year from a SAS date (1 = January, 2=February, etc.).

Syntax: MONTH (date)
date is a SAS date value.

Examples

| Function | Returns |
| :--- | :--- |
| MONTH ('16AUG2002'd) | 8 |

See Program 4.5.

## Function: WEEK

Purpose: To extract the week number of the year from a SAS date (the week-number value is a number from 0 to 53 or 1 to 53 , depending on the optional modifier).

Syntax: WEEK (<date> <,'modifier'>))
date is a SAS date value. If date is omitted, the WEEK function returns the week number of the current date.
modifier is an optional argument that determines how the week-number value is determined. If modifier is omitted, the first Sunday of the year is week 1 . For dates prior to this date, the WEEK function returns a 0 . The various modifiers provide several different methods for computing the value returned by the WEEK function. Most users will probably want to use this function without any modifiers. For details about the modifiers, see Product Documentation in the Knowledge Base, available at http://support.sas.com/documentation.

## Examples

| Function | Returns |
| :--- | :--- |
| WEEK ('16AUG2002'd) | 32 |
| WEEK ('01JAN1960'd) | 0 |
| WEEK ('03JAN1960'd) | 1 |
| WEEK ('01JAN1960'd,'V') | 53 |

See Program 4.5 for an example.

## Function: WEEKDAY

Purpose: To extract the day of the week from a SAS date ( $1=$ Sunday, $2=$ Monday, etc.).

Syntax: WEEKDAY (date)
date is a SAS date value.

## Examples

| Function | Returns |
| :--- | :--- |
| WEEKDAY ('16AUG2002'd) | 5 (Thursday) |

## Function: DAY

Purpose: To extract the day of the month from a SAS date, a number from 1 to 31 .
Syntax: DAY (date)
date is a SAS date value.
Examples

| Function | Returns |
| :--- | :--- |
| DAY ('16AUG2002'd) | 16 |

See Program 4.5.

## Program 4.5: Demonstrating the functions YEAR, QTR, MONTH, WEEK, DAY, and WEEKDAY

```
    ***Primary functions: YEAR, QTR, MONTH, WEEK, DAY, and WEEKDAY;
    data date_functions;
        set dates(drop=Date2);
        Year = year(Date1);
        Quarter = qtr(Date1);
        Month = month(Date1);
        Week = week (Date1);
        Day_of_month = day(Date1);
```

```
    Day_of_week = weekday(Date1);
run;
title "Listing of Data Set DATE FUNCTIONS";
proc print data=date_functions noobs;
run;
```


## Explanation

These basic date functions are straightforward. They all take a SAS date as the single argument and return the year, the quarter, the month, the week, the day of the month, or the day of the week. Remember that the WEEKDAY function returns the day of the week, while the DAY function returns the day of the month (it's easy to confuse these two functions). A listing of DATE_FUNCTIONS follows:

| Listing of Data Set DATE_FUNCTIONS |  |  |  |  |  |  |
| ---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Date1 | Year | Quarter | Month | Week | Day_of__ <br> month | Day_of_- <br> week |
| 01JAN1960 | 1960 | 1 | 1 | 0 | 1 | 6 |
| O2MAR1961 | 1961 | 1 | 3 | 9 | 2 | 5 |
| 25DEC2000 | 2000 | 4 | 12 | 52 | 25 | 2 |
| 01FEB2002 | 2002 | 1 | 2 | 4 | 1 | 6 |

## Functions That Extract Hours, Minutes, and Seconds from SAS Datetime and Time Values

The HOUR, MINUTE, and SECOND functions work with SAS datetime or time values in much the same way as the MONTH, YEAR, and WEEKDAY functions work with SAS date values.

## Function: HOUR

Purpose: To extract the hour from a SAS datetime or time value.
Syntax: HOUR(time or $d t$ )
time or $d t$ is a SAS time or datetime value.

## Examples

For these examples, DT = '02JAN1960:5:10:15'dt, T = '5:8:10'T.

| Function | Returns |
| :--- | :--- |
| $\operatorname{HOUR}($ DT $)$ | 5 |
| $\operatorname{HOUR}(T)$ | 5 |
| $\operatorname{HOUR}(\operatorname{HMS}(5,8,9))$ | 5 |

See Program 4.6.

## Function: MINUTE

Purpose: To extract the minute value from a SAS datetime or time value.
Syntax: MINUTE (time or $d t$ )
time or $d t$ is a SAS time or datetime value.

## Examples

For these examples, $D T=$ '02JAN1960:5:10:15'dt, $T=' 5: 8: 10 ' T$.

| Function | Returns |
| :--- | :--- |
| MINUTE (DT) | 5 |
| MINUTE (T) | 5 |
| MINUTE (HMS (5, 8, 9) ) | 5 |

See Program 4.6.

## Function: SECOND

Purpose: To extract the second value from a SAS datetime or time value.
Syntax: SECOND (time or dt)
time or $d t$ is a SAS time or datetime value.

## Examples

For these examples, DT = '02JAN1960:5:10:15'dt, T = '5:8:10'T.

| Function | Returns |
| :--- | :--- |
| SECOND (DT) | 15 |
| SECOND (T) | 10 |
| SECOND (HMS $(5,8,9))$ | 9 |

## Program 4.6: Demonstrating the HOUR, MINUTE, and SECOND functions

```
***Primary functions: HOUR, MINUTE, and SECOND;
data time;
        DT = '01jan1960:5:15:30'dt;
        T = '10:05:23't;
        Hour dt = hour(DT);
        Hour time = hour(T);
        Minute dt = minute(DT);
        Minute time = minute(T);
        Second dt = second(DT);
        Second_time = second(T);
        format DT datetime.;
run;
title "Listing of Data Set TIME";
proc print data=time noobs heading=h;
run;
```


## Explanation

The variable DT is a SAS datetime value (computed as a SAS datetime constant), and T is a SAS time value (computed as a SAS time constant). The program demonstrates that the HOUR, MINUTE, and SECOND functions can take either SAS datetime or time values as arguments. The listing follows:


## Functions That Extract the Date or Time from SAS Datetime Values

The DATEPART and TIMEPART functions extract either the date or the time from a SAS datetime value (the number of seconds from January 1, 1960).

## Function: DATEPART

Purpose: To compute a SAS date from a SAS datetime value.
Syntax: DATEPART (date-time-value)
date-time-value is a SAS datetime value.

## Examples

For these examples, DT = '02JAN1960:5:10:15'dt.

| Function | Returns |
| :--- | :--- |
| DATEPART (DT) | 1 (01JAN1960 - formatted) |
| DATEPART ('4JUN2003:20:48:15'DT) | $15860 \quad$ (04JUN2003 - formatted) |

See Program 4.7.

## Function: TIMEPART

Purpose: To extract the time part of a SAS datetime value.
Syntax: TIMEPART (date-time-value)
Date-time-value is a SAS datetime value.

## Examples

For these examples, DT = '02JAN1960:5:10:15'dt.

| Function | Returns |  |
| :--- | :--- | :--- |
| TIMEPART (DT) | $18615 \quad(5: 10: 15-$ formatted $)$ |  |
| TIMEPART ('4JUN2003:20:48:15'DT) | $74895 \quad(20: 48: 15-$ formatted) |  |

Program 4.7: Extracting the date part and time part of a SAS datetime value

```
    ***Primary functions: DATEPART and TIMEPART;
data pieces_parts;
    DT = '01jan1960:5:15:30'dt;
    Date = datepart(DT);
    Time = timepart(DT);
    format DT datetime. Time time. Date date9.;
run;
title "Listing of Data Set PIECES PARTS";
proc print data=pieces_parts noobs;
run;
```


## Explanation

The DATEPART and TIMEPART functions extract the date and the time from the datetime value, respectively. These two functions are especially useful when you import data from other sources. (In SAS 8, imported spreadsheet columns that were formatted as dates in Microsoft Excel wound up as datetime values in the SAS data set.) You can use these two functions to separate the date and time from that value. See the following listing:

```
Listing of Data Set PIECES_PARTS
\begin{tabular}{crc} 
DT & Date & Time \\
01JAN60:05:15:30 & 01JAN1960 & 5:15:30
\end{tabular}
```


## Functions That Work with Date, Datetime, and Time Intervals

Functions in this group work with date or time intervals. The INTCK function, when used with date or datetime values, can determine the number of interval boundaries crossed between two dates. When used with SAS time values, it can determine the number of hour, minute, or second boundaries between two time values.

The INTNX function, when used with SAS date or datetime values, is used to determine the date after a given number of intervals have passed. When used with SAS time values, it computes the time after a given number of time interval units have passed.

You will find an excellent description of these two functions in SAS Language Reference: Concepts or in the following technical note:

```
http://support.sas.com/techsup/technote/ts668.html
```

Note: The wording of arguments in this book might differ from the wording of arguments in the Product Documentation at http://support.sas.com.

## Function: INTCK

Purpose: To return the number of intervals between two dates, two times, or two datetime values. To be more accurate, the INTCK function counts the number of times a boundary has been crossed going from the first value to the second.

For example, if the interval is YEAR and the starting date is January 1, 2002, and the ending date is December 31, 2002, the function returns a 0 . The reason for this is that the boundary for YEAR is January 1, and even
though the starting date is on a boundary, no boundaries are crossed in going from the first date to the second. Using the same logic, going from December 31, 2002, to January 1, 2003, does cross a year boundary and returns a 1 . This is true even though, in the first case, there are 364 days between the dates and, in the latter case, only one day.

These intervals can be used "as is" or with multipliers such as two-year intervals, and they can be shifted so that the boundary is, for example, the seventh month of the year (July) instead of January 1.

When used with multi-intervals and shifted intervals, the INTCK function can become very complicated. A limited discussion of the finer points of the INTCK function follows the syntax and examples.

Syntax: INTCK('interval<Multiple><.shift>', start-value, end-value)

Intervals can be date units:

| Interval | Description |
| :--- | :--- |
| DAY | Day |
| WEEK | Week |
| WEEKDAY | Each weekday (Monday to Friday, or any <br> set of days you choose) |
| TENDAY | Ten-day period |
| SEMIMONTH | Two-week period |
| MONTH | Month |
| QTR | Quarter (Jan-Mar = 1, Apr-Jun = 2, etc.) |
| SEMIYEAR | Half year |
| YEAR | Year |

Intervals can be time units:

| Interval | Description |
| :--- | :--- |
| SECOND | Seconds |
| MINUTE | Minutes |
| HOUR | Hours |

Intervals can be datetime units:

| Interval | Description |
| :--- | :--- |
| DTDAY | Day |
| DTWEEK | Week |
| DTWEEKDAY | Each weekday (Monday to Friday) |
| DTTENDAY | Ten-day period |
| DTSEMIMONTH | Two-week period |
| DTMONTH | Month |
| DTQTR | Quarter (Jan-Mar = 1, Apr-Jun = 2, etc.) |
| DTSEMIYEAR | Half year |
| DTYEAR | Year |

interval is one item from the preceding list, placed in quotation marks.
multiple is an optional modifier in the interval. You can specify multiples of an interval. For example, MONTH2 specifies two-month intervals; DAY50 specifies 50-day intervals.
.shift is an optional parameter that determines the starting point in an interval. For example, YEAR. 4 specifies yearly intervals, starting from April 1. The shift value for single intervals is shown in the following table:

Shift value for SAS date and datetime values:

| Interval | Shift Value |
| :--- | :--- |
| YEAR | Month |
| SEMIYEAR | Month |
| QTR | Month |
| MONTH | Month |
| SEMIMONTH | Semimonth* |
| TENDAY | Tenday |
| WEEKDAY | Day |
| WEEK | Day |
| DAY | Day |

Shift value for SAS time intervals:

| Interval | Shift Value |
| :--- | :--- |
| HOUR | Hour* |
| MINUTE | Minute* |
| SECOND | Second* |

*Only multi-intervals of these intervals can be shifted.
For all multi-unit intervals except WEEK, SAS creates an interval starting from January 1, 1960. Multiple intervals are all shifted by the same unit as the non-multiple intervals (see lists above). So, YEAR4.24 specifies fouryear intervals with the interval boundary at the beginning of the second year (January 1, 1962, January 1, 1966, etc.). MONTH4.2 indicates four-month intervals, with the boundary being the first day of the second month. See the following discussion on interval multipliers and shifted intervals.

Here are some examples of intervals:

| Interval | Interpretation |
| :--- | :--- |
| YEAR | Each year |
| YEAR2 | Every two years |
| YEAR.4 | Each April |
| YEAR4.11 | November, every four years |
| MONTH | Every month |
| MONTH4 | Every four months |
| MONTH6.3 | Every six months with boundaries at |
|  | March and September |
| WEEK | Each week |
| WEEK2 | Every two weeks |
| WEEK.4 | Every week starting with |
|  | Wednesday |
| WEEK2.4 | Every two weeks starting with |
|  | Wednesday |
| WEEKDAY | Five-day weeks with weekend days, |
|  | Saturday and Sunday |
| WEEKDAY1W | Six-day weeks with weekend day, |
|  | Sunday |
| WEEKDAY12W | Five-day weeks with weekend days, |
|  | Sunday and Monday |
| HOUR | Every hour |
| HOUR4 | Every four hours |
| HOUR8.7 | Every eight hours with boundaries |
|  | 6 AM, 2 PM, and 10 PM |
| DTMONTH | Every month (used with datetime |
|  | values) |

start-value is a SAS date, time, or datetime value.
end-value is a SAS date, time, or datetime value.

## Examples

| Function | Returns |
| :--- | :--- |
| INTCK ('WEEK','16AUG2002'd,'24AUG2002'd) | 1 |
| INTCK ('YEAR', '01JAN2002'd,'31DEC2002'd) | 0 |
| INTCK ('YEAR', '01JAN2002'd,'02JAN2003'd) | 1 |
| INTCK ('YEAR', '31DEC2002'd,'01JAN2003'd) | 1 |
| INTCK ('QTR','01JAN2002'd,'01AUG2002'd) | 2 |
| INTCK ('MONTH3','01JAN2002'd,'15APR2002'd) | 1 |
| INTCK ('YEAR.7','05MAY2002'd,'15JUL2002'd) | 1 |
| INTCK ('HOUR','06:01:00't,'07:23:15't) | 1 |

See Program 4.9.

## A Discussion of Interval Multipliers and Shifted Intervals

Some applications of interval multipliers are quite straightforward. For example, if you use YEAR2 as your interval, the intervals will be every two years. The value of

INTCK ('YEAR2','15JAN2000'd,'21JAN2003'd)
is equal to 1 (one boundary, January 1, 2002, was crossed in going from January 15, 2000, to January 21, 2002). The reason that January 1, 2002, is a boundary is that the counting of boundaries goes back to January 1, 1960, which was an even number. Therefore, the boundaries will be even-numbered years.

You can shift some single intervals. For example, YEAR. 7 indicates yearly intervals with the boundary being July 1 of every year. For the intervals of YEAR, SEMIYEAR, and QTR, the shift amount is months. For example, the value of

INTCK ('YEAR. $7^{\prime},{ }^{\prime} 01$ JUN2000'd,'03JUL2002'd)
is equal to 3 (crossing boundaries at July 1, 2000, July 1, 2001, and July 1, 2002).

Shifting intervals that use multipliers is similar. For example, YEAR2.12 indicates two-year intervals, with boundaries at the second year of each interval: January 1, 1961, January 1, 1963 , etc. That is, every odd year. For example, the value of

INTCK ('YEAR2.12','15JAN2000'd,'21JAN2003'd)
is equal to 2 (crossing the boundaries at January 1, 2001, and January 1, 2003).
Multi-month intervals are shifted by months, not weeks (since there is not an even number of weeks in a month). MONTH4.2 means four-month intervals with the boundary being the second month of each four-month period. By the way, the .2 does not mean "shift the boundary by 2 months." It means the boundary is the second month of each interval. As Charley Mullin says in his technical note: "The boundary is shifted TO an interval, not BY the interval."

The value of
INTCK ('MONTH4.2','28JAN2003'd,'03JUL2003'd)
is equal to 2 (crossing the boundaries at February 1, 2003, and June 1, 2003).
WEEK and multi-week intervals present a special problem. For example, you might expect the value of

INTCK (WEEK,'01JAN1960'd,'04JAN1960'd)
to equal 0 . However, it is equal to 1 . The problem is that weekly intervals are counted every time a Sunday is crossed and January 1, 1960, is a Friday. The way that SAS decided to solve this problem was to start counting from Sunday in the same week of January 1, 1960, which is December 27, 1959. Going from January 1, 1960, to January 4, 1960, crosses a boundary (Sunday, January 3). This gets even more complicated when you are dealing with multi-week intervals.

As the default, the interval of WEEKDAY treats Saturday and Sunday as part of the preceding day. For example, the value of

INTCK ('WEEKDAY','01JUN2003'd,'30JUN2003'd)
is equal to 21. June 1, 2003, is a Sunday, and June 30 falls on a Monday. The number of times you have crossed a boundary (a working day) is 21. (Please see a further discussion of the WEEKDAY interval below the June 2009 calendar later in this chapter.)

You can specify days other than Saturday and Sunday to be treated as weekend days. For example, if you had a six-day work week, with Sunday as the day off, you could indicate the interval as WEEKDAY1W. So, the value of

INTCK('WEEKDAY1W','01JUN2003'd,'30JUN2003'd)
is equal to 25 (Monday through Saturday for four weeks plus Monday, June 30).
If you were in the restaurant business and your restaurant was closed on Sunday and Monday, you would use the interval: WEEKDAY12W to compute the number of work days between two dates.

When you are computing the number of working days between two dates, it is important to know if the starting date is a working day or not. For example, take a look at the following calendar for June 2009:

| JUNE |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S | M | T | W | T | F | S |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 |  |  |  |  |

The expression INTCK ('weekday','08jun2009'd,'12jun2009'd) returns a 4, even though there are 5 working days in the week. Remember, you are counting how many boundaries are being crossed going from date 1 to date 2 . Here, you cross a boundary on June $9,10,11$, and 12 (4 boundaries).

Now, see what happens if you start on June 7 instead of June 8. The expression INTCK ('weekday','07jun2009'd,'12jun2009'd) returns a 5, since you cross one boundary going from Sunday to Monday. By the way, if you computed the interval from June 6 to June 12, the result would also be a 5 .

Notice that it doesn't matter if the ending date is a weekday or not. For example,
INTCK('weekday','08jun2009'd,'13jun2009'd) or INTCK ('weekday','08jun2009'd,'14jun2009'd) still returns a value of 4.

## Function: INTNX

Purpose: To return the date after a specified number of intervals have passed.

## Syntax: INTNX('interval', start-date, increment <,'alignment'>)

interval is one of the same values that are used with the INTCK function (placed in quotation marks).
start-date is a SAS date.
increment is the number of intervals between the start date and the date returned by the function.
alignment is an optional argument and has a value of BEGINNING (B), MIDDLE (M), END (E), or SAMEDAY(S). The default is BEGINNING. For example, if the interval is WEEK, an increment of 1 from January 1, 1960, with the default returns the date January 3, 1960 (a Sunday, the beginning of a boundary). The same date and interval with an alignment of MIDDLE returns the date January 6, 1960 (a Wednesday, the middle of the interval).

## Examples

For these examples, DT1 = '01JAN1960:7:5:12'DT.
Note: Values in parentheses in the Returns column are the formatted values.

| Function | Returns |
| :--- | :--- |
| INTNX('WEEK','01JAN1960'd, 1) | 2 (Sunday, Jan 3, 1960) |
| INTNX('WEEK','01JAN1960'd,1,'MIDDLE') | 5 (Wednesday, Jan 6, 1960) |
| INTNX('WEEK.4','01JAN1960'd,1) | 5 (Wednesday, Jan 6, 1960) |
| INTNX('WEEK2','01JAN1960'd,1) | 9 (Sunday, Jan 10, 1960) |
| INTNX('QTR','01JAN2003'd,1) | 15796 (Tuesday, April 1, 2003) |
| INTNX('YEAR.3','01JAN2003'd,1) | 15765 (Saturday, March 1, 2003) |
| INTNX('YEAR.3','01JAN2003'd,2) | 16131 (Monday, March 1, 2004) |
| INTNX('YEAR','01JUN2003'd,1) | 16071 (Thursday, January 1, 2004) |
| INTNX('YEAR','01JUN2003'd,2) | 16437 (Saturday, January 1, 2005) |
| INTNX('YEAR4.11','01JAN2003'd,1) | 16376 (Monday, November 1, 2004) |
| INTNX('DTMONTH',DT1,3) | 7862400 (01APR60:00:00:00) |
| INTNX('HOUR','9:15:09'T,2) | 39600 (11:00:00) |
| INTNX ('YEAR','15JAN1960'D,-1) | -365 (January 1, 1959) |

## Some examples demonstrating the SAMEDAY alignment

Date = '10May2005'd (Tuesday). Return values are formatted.

| Function | Returns |  |
| :--- | :--- | :--- |
| INTNX('week', Date,1,'sameday') | 17May2005 (Tuesday) |  |
| INTNX('month', Date,1,'sameday') | 10Jun2005 (Friday) |  |
| INTNX('year', Date1,1,'sameday') | 10May2006 (Wednesday) |  |
| INTNX('weekday','7May2005'd,1, | $09 M a y 2005$ (Monday) |  |
| 'sameday') Note: this is a Saturday |  |  |

## Program 4.8: Demonstrating the INTNX function (with the SAMEDAY alignment)

```
***Primary functions: INTNX, WEEKDAY;
***Other functions: RANUNI, CEIL;
*A dentist wants to see each of his patients in six months for a followup
visit. However, if the date in six months falls on a Saturday or Sunday,
he wants to pick a random day in the following week.;
Data dental;
    input Patno : $5. Visit_date : mmddyy10.;
    format Visit date weekdate.;
datalines;
001 1/14/2009
002 1/17/2009
003 1/18/2009
004 1/19/2009
005 1/19/2009
006 1/20/2009
007 1/11/2009
008 1/17/2009
;
title "Listing of data set DENTAL";
proc print data=dental noobs;
run;
data followup;
    set dental;
    Six_months = intnx('month',Visit_date,6,'sameday');
    *Check if weekend;
    DayofWeek = weekday(six_months);
    *Keep track of actual day for testing purposes;
    Actual = Six_months;
    *If Sunday add random integer between 1 and 5;
    if DayofWeek = 1 then
        Six_months = Six_months + ceil(ranuni (0)*5);
    *If Saturday, add a random integer between 2 and 6;
    else if DayofWeek = 7 then
        Six_months = Six_months + ceil(ranuni(0)*5 + 1);
run;
title "Six Month Appointment Dates";
proc report data=followup nowd headline;
    columns Patno Visit date Actual Six months;
    define Patno / display "Patient Numb̄er" width=7;
    define Visit_date / display "Initial Date" width=15 format=weekdate.;
    define Actual / display "Actual Day" width=15 format=weekdate.;
    define Six_months / display "Six Month Appt." width=15
format=weekdate.;
run;
quit;
```


## Explanation

The introduction of the SAMEDAY alignment greatly enhanced the usefulness of the INTNX function. If you used the INTNX function in the preceding program without the SAMEDAY alignment, all of the dentist's patients would be coming in on the first of each month! Not a great plan. By using the SAMEDAY alignment, the function returns a date six months ahead, but on the same day of the month. Since this date may be a Saturday or Sunday, adjustments need to be made. In this program, it was decided that if the six month date fell on a Saturday or Sunday, a random day in the following week was to be chosen.

The expression ceil (ranuni $(0) * 5$ ) produces a random integer from 1 to 5 ; the expression ceil (ranuni $(0) * 5+1$ ) produces a random integer from 2 to 6 . For illustration purposes, the actual date six months from the visit date was not dropped from the data set so that you can see what happens if the follow-up date falls on a Saturday or Sunday. Here is the listing:

| Six Month Appointment Dates |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| Patient |  |  |  |  |
| Number | Initial Date |  | Actual Day | Six Month Appt. |
| 001 | Wed, Jan 14, 09 | Tue, | Jul 14, 09 | Tue, Jul 14, 09 |
| 002 | Sat, Jan 17, 09 | Fri, | Jul 17, 09 | Fri, Jul 17, 09 |
| 003 | Sun, Jan 18, 09 | Sat, | Jul 18, 09 | Tue, Jul 21, 09 |
| 004 | Mon, Jan 19, 09 | Sun, | Jul 19, 09 | Wed, Jul 22, 09 |
| 005 | Mon, Jan 19, 09 | Sun, | Jul 19, 09 | Fri, Jul 24, 09 |
| 006 | Tue, Jan 20, 09 | Mon, | Jul 20, 09 | Mon, Jul 20, 09 |
| 007 | Sun, Jan 11, 09 | Sat, | Jul 11, 09 | Mon, Jul 13, 09 |
| 008 | Sat, Jan 17, 09 | Fri, | Jul 17, 09 | Fri, Jul 17, 09 |

## Using the INTNX Function to Determine Starting Boundaries for Multi-Day Intervals

Interval boundaries are straightforward for intervals such as years, quarters, and months. However, suppose you want to create 12-day intervals. How many 12-day intervals are there from January 1, 2004, to January 11, 2004? How many boundaries have you crossed? The
problem here is that you have to realize that you start counting 12-day intervals from January 1,1960 , to determine where the boundaries are. Here's an easy way to see what date the counting starts on: use the INTNX function like this:

```
START_INTERVAL = INTNX('DAY12','01JAN2004'd,1)
```

The value is Saturday, January 10, 2004. So, in going from January 1, 2004, to January 11, 2004, you cross one boundary (January 10, 2004). To check, note that

INTCK ('DAY12','01JAN2004'd,'11JAN2004'd)
is equal to 1 .

## Function: YRDIF

Purpose: To return the difference in years between two dates (includes fractional parts of a year).

Important note: About the time this book was being sent to the printer, it was discovered that the YRDIF function would sometimes return a value that was off by one day for certain date intervals. It appeared this error was related to leap years. However, this author believes that, even with this error, using the YRDIF function to compute ages (or any differences in years) is still more accurate than the older method of dividing the difference in years by 365.25 . Future releases of SAS are expected to address this error with YRDIF. If you need to compute exact year differences and you are using a version of SAS that does not have the updated feature, you can use the INTCK function to accomplish your goals (see an illustration in this section).

## Syntax:

```
YRDIF(start-date, end-date, 'basis')
```

start-date is a SAS date value.
end-date is a SAS date value.
basis is an argument that controls how SAS computes the result. The first value is used to specify the number of days in a month; the second value (after the slash) is used to specify the number of days in a year.

A value of 'ACT/ACT' (alias 'ACTUAL') uses the actual number of days in a month and the actual number of days in a year (either 365 or 366 days, depending on whether there are leap years involved). For certain industries, especially financial institutions, you can specify values for the number of days in the month and the number of days in the year. This is frequently done for interest calculations on bonds and other commodities. Other choices for basis are:
'30/360' Uses 30-day months and 360-day years in the calculation.
'ACT/365' Uses the actual number of days between the two dates, but uses 365-day years, even if a leap year is in the interval.
'ACT/360' Uses the actual number of days between the two dates, but uses 360 -day years.

## Examples

| Function | Returns |
| :--- | :--- | :--- |
| YRDIF ('01JAN2002'd,'01JAN2003'd,'ACTUAL') | 1 |
| YRDIF ('01JAN2002'd, 01FEB2002'd,'ACT/ACT') | .0849 |
| YRDIF ('01FEB2002'd,01MAR2003'd,'ACTUAL') | 1.9767 |
| YRDIF ('01JAN2002'd,'01JAN2003'd,'ACT/365') | 1.0139 |

Program 4.9: Program to demonstrate the date interval functions

```
    ***Primary functions: INTCK, INTNX, YRDIF;
    data period;
        set dates;
        Interval_month = intck('month',Date1,Date2);
        Interval year = intck('year',Date1,Date2);
        Year_diff}=\mathrm{ yrdif(Date1,Date2,'actual');
        Interval_qtr = intck('qtr',Date1,Date2);
        Next_month = intnx('month',Date1,1);
        Next_year = intnx('year',Date1,1);
        Next_qtr = intnx('qtr',Date1,1);
        Six_month = intnx('month',Datel,6);
        format Next: Six_month date9.;
    run;
```

```
title "Listing of Data Set PERIOD";
proc print data=period heading=h;
    id date1 date2;
run;
```


## Explanation

Before we discuss the date functions in this program, let me point out that the ID statement of PROC PRINT lists both DATE1 and DATE2 as ID variables. This allows the values to be repeated on the lower portion of the listing.

The interval functions can be somewhat confusing. It helps to keep in mind that the INTCK function counts how many times you cross a boundary going from the start date to the end date. The listing follows:

| Listing of Data Set PERIOD |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Date1 Date2 | $\begin{aligned} & \text { Interval_ } \\ & \text { month } \end{aligned}$ | Interval year | Year_ diff | Interval qtr | Next_ month |
| 01JAN1960 15JAN1960 | 0 | 0 | 0.03825 | 50 | 01FEB1960 |
| 02MAR1961 18FEB1962 | 11 | 1 | 0.96712 | 24 | 01APR1961 |
| 25DEC2000 03JAN2001 | 1 | 1 | 0.02461 | 1 | 01JAN2001 |
| 01FEB2002 31MAR2002 | 1 | 0 | 0.15890 | 0 | 01 MAR2002 |
| Date1 Date2 | Next_year | Next_q | qtr S | Six_month |  |
| 01JAN1960 15JAN1960 | 01JAN1961 | 01APR19 | 960 0 | 01JUL1960 |  |
| 02MAR1961 18FEB1962 | 01JAN1962 | 01APR19 | 961 0 | 01 SEP1961 |  |
| 25DEC2000 03JAN2001 | 01JAN2001 | 01JAN20 | 001 0 | 01JUN2001 |  |
| 01FEB2002 31MAR2002 | 01JAN2003 | 01 APR20 | 0020 | 01AUG2002 |  |

## Computing Exact Ages

If you are using a release of SAS that has not corrected the possible error in the YRDIF function, you can use the following SAS statement to compute AGE exactly (submitted by my friend Mike Zdeb):

```
Age_exact = floor((intck('month',DOB,Date)-(day(Date) < day(DOB))) / 12);
```


# Function That Computes Dates of Standard Holidays 

## Function: HOLIDAY

Purpose: Returns a SAS date, given a holiday name and a year.
Syntax: HOLIDAY (holiday, year)
holiday is a holiday name (see list below).
year is a numeric variable or constant that represents the year.

## Partial List of Holidays:

| Christmas | Christmas day (December 25) |
| :--- | :--- |
| Columbus | Columbus day (2nd Monday in October) |
| Easter | Easter Sunday |
| Fathers | Father's Day (3rd Sunday in June) |
| Halloween | Halloween |
| Labor | Labor Day (1st Monday in September) |
| MLK | Martin Luther King Day (celebrated on |
|  | Monday) |
| Memorial | Memorial Day (1st Monday in May) |
| Mothers | Mother's Day (2nd Sunday in May) |
| Newyear | New Year's Day (January 1) |
| Thanksgiving | Thanksgiving (4th Thursday in November) |
| USIndependence | July 4th holiday |
| USPresidents | President's Day (3rd Monday in February) |
| Veterans | Veterans Day (November 11) |
| VeteransUSG | Veterans Day (U.S. Government) |

## Examples

| Function | Returns |  |
| :--- | :--- | :--- |
| HOLIDAY('Christmas',2009) | $12 / 25 / 2009$ (Friday) |  |
| HOLIDAY('USIndependence',2009) | $7 / 4 / 2009$ (Saturday) |  |
| HOLIDAY('VeteransUSG', 2009) | $11 / 11 / 2009$ (Wednesday) |  |
| HOLIDAY('Easter',2009) | $4 / 12 / 2009$ (Sunday) |  |
| HOLIDAY('MLK',2009) | $1 / 19 / 2009$ (Monday) |  |
| HOLIDAY('Thanksgiving',2009) | $11 / 26 / 2009$ (Thursday) |  |

## Program 4.10: Demonstrating the HOLIDAY function

***Primary function: HOLIDAY;
***Other functions: WEEKDAY, INTCK;
data salary;
H1 = holiday('Newyear', 2005);
if weekday(H1) = 7 then H1 = H1 + 2;
else if weekday (H1) = 1 then H1 = H1 + 1;
H2 = holiday('MLK',2005);
H3 = holiday('USpresidents',2005);
H4 = holiday('Easter',2005)-2;
array H[4];
First = '01Jan2005'd; *Saturday;
Second = '31Mar2005'd; *Thursday;
Work = intck('weekday',First,Second);
/* if holiday falls between the First and Second date, decrement number of working days */
do i = 1 to 4;
if First le H[i] le Second then Work = Work - 1;
end;
Salary = 500 * Work;
format First Second mmddyy10. Salary dollar10.;
keep First Second Work Salary;
run;
title "Listing of SALARY";
proc print data=SALARY noobs;
run;

## Explanation

In this program, you want to compute the number of working days between January 1, 2005 (Saturday) and March 31, 2005.

The INTCK function with the WEEKDAY interval computes the number of times you cross working day boundaries going from one date to another (with Monday through Friday being defined as the default working days).

Note: Since the starting date is a Saturday, you do not have to add one to the value returned, because going from a weekend day to Monday crosses a boundary. If the starting date were not on a weekend, you would need to add one to the variable WORK. (Please see the explanation following the June 2009 calendar earlier in this chapter.)

Next, you want to test if any of the holidays (New Year's Day, Martin Luther King's birthday, President's Day, or Easter) fall in that interval. In addition, since New Year's Day can fall on any day of the week, you use the WEEKDAY function to test if this holiday falls on a Saturday or Sunday. If so, your company gives its employees the following Monday off. Easter always falls on a Sunday, so employees are given the previous Friday (Good Friday) off.

An array is created to hold the four non-working days. Finally, you test if each of the four non-working days fall in the given interval. If so, you decrement the number of working days computed by the INTCK function.

Here is the listing:

| Listing of SALARY |  |  |  |
| ---: | ---: | ---: | ---: |
| First | Second | Work | Salary |
|  |  |  |  |
| $01 / 01 / 2005$ | $03 / 31 / 2005$ | 60 | $\$ 30,000$ |

## Functions That Work with Julian Dates

This group of functions involves Julian dates. Julian dates are commonly used in computer applications and represent a date as a two- or four-digit year followed by a three-digit day of the year ( 1 to 365 or 366 , if it is a leap year). For example, January 3, 2003, in Julian notation would be either 2003003 or 03003 . December 31, 2003 (a non-leap year) would be either 2003365 or 03365.

## Function: DATEJUL

Purpose: To convert a Julian date into a SAS date.
Syntax: DATEJUL(jul-date)
jul-date is a numerical value representing the Julian date in the form dddyy or dddyyyy.

## Examples

For these examples, JDATE = 1960123.

| Function | Returns |
| :--- | :--- |
| DATEJUL (1960001) | $0 \quad(01 J A N 1960$ formatted $)$ |
| DATEJUL (2003365) | $16070 \quad(31 D E C 2003$ formatted $)$ |
| DATEJUL (JDATE) | $122 \quad(02 M A Y 1960$ formatted $)$ |

See Program 4.11.

## Function: JULDATE

Purpose: To convert a SAS date into a Julian date.
Syntax: JULDATE (date)
date is a SAS date.

## Examples

For these examples, DATE = '31DEC2003'D.

| Function | Returns |
| :--- | :--- |
| JULDATE (DATE) | 3365 |
| JULDATE ('01 JAN1960'D) | 60001 |
| JULDATE (122) | 60123 |

See Program 4.11.

## Function: JULDATE7

Purpose: To convert a SAS date into seven-digit Julian date.
Syntax: JULDATE7 (date)
date is a SAS date.

## Examples

For these examples, DATE = '31DEC2003'D.

| Function | Returns |
| :--- | :--- |
| JULDATE7 (DATE) | 2003365 |
| JULDATE7 ('01JAN1960 ' D) | 1960001 |
| JULDATE7 (122) | 1960123 |

Program 4.11: Demonstrating the three Julian date functions

```
    ***Primary functions: DATEJUL, JULDATE, and JULDATE7.;
***Note: option YEARCUTOFF set to 1920;
options yearcutoff = 1920;
data julian;
    input Date : date9. Jdate;
    Jdate_to_sas = datejul(Jdate);
    Sas_to_Jdate = juldate(Date);
    Sas_to_jdate7 = juldate7(Date);
    format Date Jdate_to_sas mmddyy10.;
datalines;
01JAN1960 2003365
15MAY1901 1905001
210CT1946 5001
;
title "Listing of Data Set JULIAN";
proc print data=julian noobs;
    var Date Sas_to_jdate Sas_to_jdate7 Jdate Jdate_to_sas;
run;
```


## Explanation

It is important to realize that Julian dates without four-digit years will be converted to SAS dates, based on the value of the YEARCUTOFF system option. To avoid any problems, it is best to use seven-digit Julian dates. The listing is shown next:

| Listing of Data Set JULIAN |  |  |  |  |
| ---: | ---: | ---: | ---: | ---: |
|  | Sas_to_ <br> Jdate | Sas_to_- <br> jdate | Jdate | Jdate__ <br> to_sas |
|  |  |  |  |  |
| $01 / 01 / 1960$ | 60001 | 1960001 | 2003365 | $12 / 31 / 2003$ |
| $05 / 15 / 1901$ | 1901135 | 1901135 | 1905001 | $01 / 01 / 1905$ |
| $10 / 21 / 1946$ | 46294 | 1946294 | 5001 | $01 / 01 / 2005$ |

From $S A S^{\circledR}$ Functions by Example, Second Edition. Full book available for purchase here.

## Index

## A

abbreviations, converting words to 110-111
ABS function 266, 273-274
address lists, locating ZIP codes in 157-158
AGE format 403
ages
computing 186, 211, 228-230
grouping 232-233
alphabetical listings by last name 101-102
ampersand (\&) 346, 368
AND function 411,413
ANYALNUM function 17-18
ANYALPHA function $18-19,24$
ANYDIGIT function 19-20, 24-25
ANYDTDTE10. informat 44
ANYDTDTE informat 327
ANYLOWER function 23
ANYPUNCT function 20-21
ANYSPACE function 21-22, 25
ANYUPPER function 22
ARCOS function 355
area codes, extracting from phone numbers
161-162
array functions
background information 219
computing number of variables in data sets 222
determining array bounds 223-224
setting missing values $220-221$
array references, as arguments to CALL routines 176
ARRAY statement
DIM function and 221
RANK function and 132
replacing numeric with missing values 397
SET statement 221
SUBSTR function and 49
arrays
determining number of elements in 219-220
multidimensional 223
sort functions and 173, 175-176
ARSIN function 356-357
asterisk (*) 219, 221
ATAN function 357-358
ATTRC function
data set ID and 376, 380
displaying data set attributes 387
functionality 375-376
attributes, data set 387-389
ATTRN function
data set ID and 377, 380
displaying data set attributes 387
functionality 375, 377-379
average, moving 317
averaging test scores 253-255

## B

backwards writing 135-136
BAND function 413, 415
Bernouilli distribution 312
BEST32. format 359
BINARY4. informat 416
BINARY8. informat 416
binomial distribution 312
bitwise logical functions 411-420
blanks
CALL routines handling 57-58
converting multiple to single $11-12$
removing from strings 70-78
blood pressure, computing changes in 318-321
BNOT function 412-413, 415
BOR function 414-415
bubble sort algorithm 262
BXOR function
background information 411

BXOR function (continued)
enciphering text using keys 416-417
encrypting and decrypting macros 418-420
functionality 414-415
BYTE function 420

## C

CALL CATS routine 58-59,61
CALL CATT routine 59, 61
CALL CATX routine 60-61
CALL COMPCOST routine
COMPGED function and 88-89
functionality 81-82
fuzzy matching and 89
CALL EXECUTE routine 365-368
CALL MISSING routine 333-334
CALL PRXCHANGE routine 168-170
CALL PRXFREE routine 171-172
CALL PRXNEXT routine 164-166
CALL PRXPOSN routine 160-163
CALL PRXSUBSTR routine 156-159
CALL RANNOR routine 308-310
CALL RANUNI routine 302
CALL routines
array references as arguments to 176
concatenating strings $57-58,61$
DO loops and 105
handling blanks 57-58
LENGTH function and 58
LENGTH statement and 58-60
Perl regular expressions and 155
PRXPARSE function and 150
random functions and 288-290
regular expressions and 155
sorting values 173-177
CALL SCAN routine 102-106
CALL SORTC routine
background information 173
constants and 176
functionality 176-177
sorting values within observations 261

CALL SORTN routine
background information 173
computing grades by dropping lowest scores 253, 259
functionality $174-176$
sorting values within observations 261, 263
CALL STREAMINIT routine 311, 313
CALL SYMPUT routine 359-363, 379
CALL SYMPUTX routine 359-363, 397
CALL VNEXT routine 399-401
capitalizing letters in strings $5-10$
capture buffers $147,160,170$
Cartesian product 91, 96
case
See also UPCASE function
changing 5-10
ignoring 96
CAT function
concatenation operator $(\|)$ and 62
COUNTC function and 69
differences among routines 57,68
FINDC function and 69
functionality 62-63
LENGTH statement and 62-65, 67
CATQ function 57, 66-67, 69
CATS function 57, 63-64, 68-69, 369
CATT function 57, 64-65, 68
CATX function 10, 57, 65-66, 69
CEIL function
demonstrating 233-234
functionality $122,226-227$
INT function and 227
simulating random dice throws 293
CEILZ function 225
CHAR function 54-56
\$CHAR. informat 70-71, 86
\$CHAR3. informat 172
\$CHAR10. informat 80
\$CHAR15. format 404
\$CHAR15. informat 71, 406
character functions
changing case 5-10
comparing strings 78-97
computing length of strings 111-114
counting letters or substrings in strings 114-121
dividing strings into words $97-106$
extracting parts of strings 46-57
joining multiple strings 10, 57-69
miscellaneous string functions 121142
removing blanks from strings 70-78
removing characters from strings
11-16
searching for characters 16-46
storage length of character variables 3-5
substituting letters or words in strings 106-111
CHARACTER_keyword 221-222
character-to-numeric conversion 14-15, 321-332
character values
setting to missing 220-221
sorting 176-177
variables and 322
character variables
assigning values at run time 330-332
character values and 322
computing number of 222
determining 392-403
formats and 404
informats and 406
storage length of 3-5, 111, 395-396
variable information in data sets 400-401
CHART procedure 307
\$CHARw. informat 71
chi-square with Yates' correction 274
CHOOSEC function 121-123
CHOOSEN function 123-125
CIMPORT procedure 348
cleansing data 171-172
CLOSE function
data set ID and 375
determining variable name 386
functionality 375,379
COALESCE function 284-285
COALESCEC function 128-129, 284
colon (:) 44, 224
COMMA11. informat 15
COMPARE function 78-81
comparing
constants 234-236
strings 78-97
comparison operator (:=) 80
COMPBL function 11-12
COMPGED function
CALL COMPCOST routine and 88-89
demonstrating 86-87
functionality 81-84
fuzzy matching and 89
generalized edit distance method and 82
ignoring cases 96
n-literals and 83
SPEDIS function and 92
VALIDVARNAME system option and 84
COMPLEV function
demonstrating 86-87
functionality $81,84-85$
fuzzy matching and 89
ignoring cases 96
Levenshtein edit distance method and 84
n -literals and 85
SPEDIS function and 92
COMPRESS function
converting numeric values of mixed units 37
extracting phone numbers from strings 159
functionality $12-16$
concatenating multiple strings 10, 57-69
concatenation operator (||)
CALL routines and 57-58, 61
CAT function and 62
concatenation operator (||) (continued)
TRIM function and $57,61,75$
conditional SET statement 222
CONSTANT function 268-270, 354
constants
CALL SORTC routine and 176
comparing 234-236
computing 268-270
Euler's constant 268
machine 268-269
machine precision 268
mathematical 268-269
quotation marks and 4
CONTENTS procedure
file I/O functions and 371
LENGTH functions and 111
NVALID function and 141
storage length of character variables 4-5
VARFMT function and 381
VARLABEL function and 382
VARLEN function and 383
VARNUM option 4
VARTYPE function and 386,388
converting
character cases 5-10
character-to-numeric 14-15, 321-332
date values 214-217
FIPS codes 336-338, 339, 341, 342
mixed numbers to decimals 100
multiple blanks to single 11-12
numeric-to-character 321-332
numeric values of mixed units 36-37
Social Security numbers 14-15
state codes 337-341, 344
words to abbreviations 110-111
ZIP codes 341-346
COS function 352-353
\$COST format 328
COUNT function 114-116
COUNTC function
CAT functions and 69
functionality 114, 116-118
modifiers and 116-117
counting
letters or substrings in strings 114-121
number of digits in character strings 15-16
words in a string 105-106
COUNTW function
counting words in a string 105
delimiters and 118-119
demonstrating 120-121
functionality $114,118-120$
modifiers and 119-120
D
dash (-) 13-14, 44
data cleansing 171-172
data set ID (DSID)
ATTRC function and 376, 380
ATTRN function and 377,380
CLOSE function and 375
defined 374
DSNAME function and 379-381
OPEN function and 374-375
data sets
choosing every $n$th observation
271-272
closing 380
computing number of variables in 222-223
creating 187, 372-373, 393-394
determining existence of 373-374
determining variable information in 400-401
displaying attributes 387-389
listing contents 255
returning characteristics of 375-381
DATA step
closing data sets 380
executing macros 365-368
freeing resources 171
obtaining macro variable values 368369
passing values $362-365$
storage length of character variables 3-4
DATALINES statement 250
DATE function 180, 184-185
date values
computing holidays 212-214
computing spread of 244-247
converting 214-217
creating 180-188
defined 180
demonstrating functions 187-188
determining order of 247-266
extracting parts of 188-192
Julian dates 214-217
reading in mixed formats $43-44$, 326-327
working with intervals 197-211
DATE9. informat 44, 187
DATEJUL function 214-215
DATEPART function 195-196
DATETIME function 180, 185
datetime values
creating 180-188
defined 180
extracting parts of 192-197
working with intervals 197-211
DAY function 191-192
decimals, converting mixed numbers to 100
deciphering text 416-418
decrypting macros 418-420
delimiters
CALL SCAN routine and 103
COUNTW function and 118-119
FINDW function and 39
reading tab-delimited files 101-102
SCAN function and 98
slashes as 145,150
descriptive statistics functions
background information 238
computing grades based on highest scores 252
computing grades by dropping lowest scores 258-260
computing lowest golf scores 250-251
computing spread of data values 244-247
computing sums, means, medians 241-244
determining number of missing values in list of variables 238-241
determining order of data values 247-266
performing $t$-test 264-266
DHMS function 183
dice, simulating random throws 293
DIF function 316, 318-321
DIFn function 316, 318-321
digits, finding in random positions in strings 165
DIM function
ARRAY statement and 221
computing numbers of variables in data sets 222
enciphering text using keys 416
encrypting and decrypting macros 419
functionality 219-220
multidimensional arrays and 223
setting numeric values to missing 220-221
disitance
between ZIP codes 335, 347-348
geographical 346-350
dividing strings into words 97-106
DLM option, INFILE statement 101
DO loops
CALL routines and 105
processing array elements 219,221
RANK function and 133
sorting values within observations 261-262
SUBSTR function and 49
dollar sign (\$) 404, 406
DOLLAR8.2 format 404
DOLLAR8.2 informat 406

## 434 Index

DSID
See data set ID
DSNAME function 379-381

## E

ELSE IF statement 232
enciphering text 416-418
encrypting macros 418-420
END= option
INFILE statement 172
SET statement 126
Euler's constant 268
EXIST function
determining number of observations in data sets 379
displaying data set attributes 387
functionality 373-374
EXP function 276-279
exponentiation 276-279
expressions
See also Perl regular expressions
logical 129-131
macro 363-365
regular expressions 143-149
extracting
area codes from phone numbers 161-162
parts of date values 188-192
parts of datetime values 192-197
parts of strings 46-57
parts of time values 192-195
phone numbers from strings 159

## F

FACT function 279-280
factorial and gamma functions 279-282
false positives 156
Federal Information Processing Standards codes
See FIPS codes
\$15. informat 71
file I/O functions
background information 371

CONTENTS procedure and 371
creating test data sets 372-373
determining existence of data sets
373-374
opening and closing files 374-375
returning data set characteristics
375-381
returning variable information 381-389
FIND function
functionality 32-33
INDEX function and 41
PRXMATCH function and 146
searching strings and characters 35-36
FINDC function
CAT functions and 69
functionality 34-39
INDEXC function and 32, 41
modifiers and 38-39
FINDW function 32, 39-41
FIPNAME function 336, 338
FIPNAMEL function 336-338
FIPS codes
converting from state codes 339,341
converting from ZIP codes 342
converting to state codes 337
converting to state names 336-338
defined 335
FIPSTATE function 337-338
FIRST function 55-57
FIRST. variable 319
FIRSTOBS $=$ system option 378
FLOOR function
demonstrating 233-234
functionality 227-228
INT function and 230
FLOORZ function 225
FORMAT statement 224
formats
See also specific formats
character variables and 404
dollar sign and 404
functions determining information 403-405
numeric variables and 404
forward slash (/)
as delimiter 145,150
in date formats 44
in regular expressions 145,147
freeing resources 171
FREQ procedure 293, 306
functions
See also character functions
See also descriptive statistics functions
See also specific functions
array 219-224
bitwise logical 411-420
determining format information 403-405
determining informat information 406-410
factorial and gamma 279-282
file I/O 371, 372-389
macro 359-369
mathematical 267, 268-285
random number 122, 287-313
sort 174-177
trigonometric 351-358
truncation 225, 226-236
variable information 391-410
fuzzing results 225
fuzzy matching $81,89-92,95-97$

## G

GAMMA function 279, 281-282
gamma functions 279-282
\$GENDER format 403
generalized edit distance method
COMPGED function and 82
defined 81
demonstrating 86-88
GEODIST function $335,346-347$, 349-350
geographical distance, computing 346-350
golf scores, computing lowest 250-251
GPLOT procedure 282
grades
averaging test scores 253-255
computing based on highest scores 252
computing by dropping lowest scores
253, 258-260
grading quizzes $174-175$
rounding 231-232
groups
grouping ages 232-233
placing values into $225,232-233$
randomly assigning groups 298-302
randomly assigning subjects 294-302
GROUPS $=$ option, RANK procedure 296

## H

HBOUND function 223-224
HMS function 184
HOLIDAY function 180, 212-214
HOUR function 192-193, 195
hyphen (-) 13-14, 44

## I

I/O functions
See file I/O functions
ID statement, PRINT procedure 211
IF statement 232
IF-THEN statement 354
IFC function 129-131
IFN function 282-283
importing data from spreadsheets 196
INDEX function 32, 41-42
INDEXC function 32, 41-44
INDEXW function 32, 41
INFILE statement
DATALINES statement and 250
DLM option 101
END= option 172
MISSOVER option 74, 250
INFORMAT statement 187, 409
informats
See also specific informats
ampersand modifiers 346
assigning at run time 324-326
character variables and 406

## 436 Index

informats (continued)
dollar sign and 406
functions determining information 406-410
numeric variables and 406
INPUT function
?? modifier 323
converting character values to numeric 108, 322-323
converting mixed numbers to decimals 100
converting SSN to numeric form 15
creating data sets 187
extracting portions of character values 47
finding digits in random positions in strings 165
finding first numbers in strings 26
functionality 321-322
PUT function and 321
reading unstructured data 163
unpacking strings 49
INPUTC function 324-325, 329
INPUTN function 324, 326-327, 329
INT function
CEIL function and 227
demonstrating 233-234
FLOOR function and 230
functionality $186,229-230$
INTCK function
computing holidays 213-214
demonstrating 210-211
functionality 197-198
intervals supported 198-205, 213
YRDIF function and 209
intervals 197-211
INTCK function and 198-205, 213
INTNX function and 206, 208-209
multi-day 208-209
multipliers for 202-205
shifted 202-205
YRDIF function and 210
INTNX function
demonstrating 206-208, 210
functionality 205
intervals supported 206, 208-209
INTZ function 225
inverse trigonometric functions 355-358
IQR function 245-246

## J

joining multiple strings $10,57-69$
JULDATE function 215-216
JULDATE7 function 216-217
Julian dates 214-217

## K

keys, enciphering text with 416-417

## L

1 modifer 99
LAG function 315-317, 320-321
LAGn function 316-317
LARGEST function
computing average test scores 253-254
computing grades by dropping lowest scores 259
functionality 251-253
missing values and 257
last name, alphabetical listings by 101-102
LAST. variable 319
LBOUND function 223-224
LEFT function 57, 61, 70-71
length
of character variables 3-5, 111
of strings, computing 111-114
storage length of character variables 3-5, 111, 395-396
LENGTH function
CALL routines and 58
CONTENTS procedure and 111
demonstrating 113-114
differences among functions 16,111 , 114
finding digits in random positions in strings 165
functionality 111
macro variables and 111
SUBSTR function and 48,51
SUBSTRN function and 53-54
VLENGTH function and 396
LENGTH statement
CALL routines and 58-60
CALL VNEXT routine and 400
CAT functions and 62-65, 67
CHAR function and 55
COALESCEC function and 129
determining variable attributes 384
replacing numeric with missing values 397
SUBSTR function and 51
SUBSTRN function and 53-54
LENGTHC function 76, 111, 112, 113-114
LENGTHM function 111, 112, 113-114
LENGTHN function 16, 111, 113-114
letters
changing case of 5-10
counting in strings 114-121
substituting in strings 106-111
Levenshtein edit distance method
COMPLEV function and 84
defined 81
demonstrating 86-88
lines with exact text matches 145-147
listings, alphabetical by last name 101-102
literals
See n-literals
LOG function 277-279
LOG10 function 277-279
logarithms 276-279
logical expressions, evaluating 129-131
LOWCASE function 7-9
\$LOWCASE informat 8
lowercase conversions 5-10

## M

machine constants 268-269
machine precision constants 268
macro expressions 363-365
macro functions 359-369
macro variables
assigning values 360-363
LENGTH functions and 111
macro expressions and 363
obtaining values of 368-369
passing values to 364-365
\&SYSDAY automatic macro variable 366
macros
averaging test scores 253-255
computing number of variables in data sets 222
displaying data set attributes 387-389
encrypting and decrypting 418-420
executing 365-368
macro expressions and 363
randomly assigning groups 298-302
replacing specific values 398-399
sorting values within observations
263-264
matching Social Security numbers 94-95
mathematical constants 268-269
mathematical functions
background information 267
commonly used 268-276
factorial and gamma functions 279-282
for exponentiation and logarithms 276-279
miscellaneous 282-285
MAX function 248-249, 257
MDY function 180-183
mean, computing 241-244
MEAN function
background information 238
computing quiz scores 259
example 240, 243
functionality 241
performing $t$-test 266
MEANS procedure 305, 366
median, computing 241-244
MEDIAN function 242-243

## 438 Index

memory, releasing 171-172
Mersenne-Twister random number generator 310
metacharacters 147-149
Microsoft Excel 196
MIN function
computing average scores 253-254
computing quiz scores 260-261
functionality 247
missing values and 257
SMALLEST function and 249
MINUTE function 192-193, 195
MISSING function 125-128, 387
missing values
determining in list of numeric values 239-241
determining in list of variables 238241
LARGEST function and 257
MIN function and 257
ORDINAL function and 257
replacing numeric values with 396-398
setting 220-221
setting arguments to 333-334
SMALLEST function and 249
MISSOVER option, INFILE statement 74, 250
mixed units, converting numeric values of 37
MMDDYY10. format 322
MMDDYY10. informat 44
MOD function
as toggle switch 272-273
background information 267
choosing every $n$th observation 271-272
functionality 270-271
modifiers
?? modifier 323
ampersand modifier 346
CALL SCAN routine and 103
COUNTC function and 116-117
COUNTW function and 119-120

FINDC function and 38-39
for informats 346
INPUT function and 323
p modifier 99
PRXPARSE function and 150
SCAN function and 98-99, 101
Monte Carlo simulation 304-307
MONTH function 189, 191-192
moving average, computing 317
multi-day intervals 208-209
multidimensional arrays 223

## N

N_automatic variable 144, 266
N function
computing average test scores 253-254
computing quiz scores 260-261
functionality 238-239
performing $t$-test 266
n-literals
COMPARE function and 78
COMPGED function and 83
COMPLEV function and 85
defined 79, 85
NLITERAL function 136-138
NMISS function 239-241
NOBS $=$ option, SET statement 222
normal distribution 303-310, 312
NOT function 411
NOTALNUM function 26-27
NOTALPHA function 27-28
NOTDIGIT function 28-29, 31
NOTLOWER function 30-31
NOTUPPER function 29-30
_NUMERIC_keyword 221-222
numeric-to-character conversion 321-332
numeric values
computing spread of 244-245
converting numeric values of mixed units 37
determining largest 248-249, 251-253
determining number of missing values 239-241
determining smallest 247-251
replacing with missing values 396-398
returning 234-236
rounding and truncating 226-234
setting to missing 220-221
sorting 174-176, 261-264
variables and 322
numeric variables
computing number of 222
determining 392-403
formats and 404
informats and 406
setting values in 220-221
storage length of 395-396
variable information in data sets 400-401
NVALID function 138-142
NYSIIS algorithm 89

OBS= system option 378
observations, choosing every $n$th 271-272
ODS (Output Delivery System) 304, 306
ODS TRACE statement 306
OPEN function
data set ID and 374-375
determining variable name 386
displaying data set attributes 387
functionality 374-375, 386
variable information functions and 391
operators
See also concatenation operator (||)
comparison operator (:=) 80
repetition operators 147,149
OR function 411, 414
ORDINAL function
background information 173
computing quiz scores 258-261
functionality 257-258
missing values and 257
sorting values within observations 261-264
Output Delivery System (ODS) 304, 306

## P

p modifer 99
$p$-values 304, 306
PAGESIZE option, REPORT procedure 354
PANELS option, REPORT procedure 275-276, 309, 354
parentheses ()
array elements and 219
regular expressions and 147
removing from phone numbers 13-14
PCTL function 255-257
Perl regular expressions
ANY functions and 24
background information 143-144
CALL routines and 155
defining 150-151
extracting area codes from phone numbers 161-162
extracting phone numbers from strings 159
finding digits in random positions in strings 165-166
locating lines with exact text matches 145-147
locating text patterns 152-167
locating ZIP codes in address lists 157-158
reading unstructured data 162-163
releasing memory used by 171-172
searching for phone numbers in strings 153-155
substituting one string for another 168170
tutorial on 144-149
phone numbers
extracting area code from 161-162
extracting from strings 159
removing dashes and parentheses 13-14
searching for in strings 153-155
Poisson distribution 312
pound sign (\#) 104

## 440 Index

PRINT procedure
ID statement 211
listing data set contents 255
\$QUOTE format 71, 73
probability distributions 287, 310-313
PROBCHI function 274
PROBT function 266
PROPCASE function 9-10, 96
PRX functions 143-144, 149, 155
PRXCHANGE function 145
PRXMATCH function
CALL PRXPOSN routine and 160-161
CALL PRXSUBSTR routine and 156
extracting phone numbers from strings 159
FIND function and 146
functionality 152-156
PRXPAREN function and 166
PRXPARSE function and 152-153, 155-156
tutorial on 145-146
PRXNEXT function 145, 163
PRXPAREN function 145, 166-167
PRXPARSE function
CALL PRXCHANGE routine and 168-169
CALL PRXNEXT routine and 164
CALL PRXPOSN routine and 160
CALL PRXSUBSTR routine and 156
CALL routines and 150
data cleaning example 171-172
functionality 150-151
modifiers and 150
PRXMATCH function and 152-153, 155-156
PRXPAREN function and 166
RETAIN statement and 150
tutorial on 144-146
PRXPOSN function 145, 147, 166
PRXSUBSTR function 52, 145
PUT function
assigning values at run time 330
functionality 327-328
INPUT function and 321
INPUTN function and 327
performing table look-ups 328
PUTC function and 329
ROUND function and 232
TRANSLATE function and 107
VVALUE function and 327, 402
PUTC function 329-331
PUTN function 329, 331-332

## Q

QTR function 189, 191-192
quizzes
computing scores 258-261
grading 174-175
quotation marks
?? modifier 323
character constants and 4
\$QUOTE format and 71,73
\$QUOTE format 71, 73

## R

RAND function 287, 310-313
random dice throws, simulating 293
random number functions
background information 122, 287-288
CALL routines and 288-290
generating normally distributed numbers 303-310
generating uniform random numbers 290-302
probability distributions and 310-313
random number seeds 288-290
random samples 291-292
randomly assigning groups 298-302
randomly assigning subjects 294-302
RANGE function 244, 246
RANK function
ARRAY statement and 132
DO loops and 133
enciphering text using keys 416-417
encrypting and decrypting macros 419-420
functionality 132-133
RANK procedure 296
RANNOR function 303-307, 310
RANUNI function
background information 122
encrypting and decrypting macros 418-420
functionality 290-291
RAND function and 310
randomly assigning groups 294-302
selecting random samples 291-292
simulating random dice throws 292294
reading
date values in mixed formats 43-44, 326-327
tab-delimited files 101-102
unstructured data 162-163
regular expressions
See also Perl regular expressions
background information 143-144
CALL routines and 155
creating 144
tutorial on 144-149
wild cards and 147-149
releasing memory 171-172
removing
blanks from strings 70-78
dashes from phone numbers 13-14
parentheses from phone numbers 13-14
specified characters from character values 12-16
RENAME $=$ data set option 322
REPEAT function 5, 133-135
repetition operators 147, 149
REPORT procedure
PAGESIZE option 354
PANELS option 275-276, 309, 354
RESOLVE function 363-365
resources, freeing 171
RETAIN statement
enciphering text using keys 417
${ }^{\mathrm{N}}$ _ automatic variable and 144
$\overline{\mathrm{P}} \mathrm{R} \overline{\mathrm{X}}$ PARSE function and 150
trigonometric functions and 354
REVERSE function 135-136
RIGHT function 72-73
Rogers, Will 147
ROUND function
demonstrating 233-234
functionality $37,230-232$
placing values into groups 225 , 232-233
PUT function and 232
simulating random dice throws 293
rounding numerical values 226-234
ROUNDZ function 225
RX functions 143-144

## S

SCAN function 97-102, 182
SCANQ function 97
searching
for characters 16-46
for phone numbers in strings 153-155
strings and characters 35-36
SECOND function 192, 194-195
SET statement
ARRAY statement and 221
CALL VNEXT routine and 401
conditional 222
END= option 126
NOBS $=$ option 222
shifted intervals 202-205
SIN function 352-353
slashes as delimiters 145,150
SMALLEST function 249-251, 257
Social Security numbers
converting 14-15
matching 94-95
sort algorithm, bubble 262
sort functions
array elements as arguments 173, 175-176
grading quizzes 174-175
sorting character values 176-177
sorting numeric values 174-176
sorting values 173-177
character values 176-177
within observations 261-264
SOUNDEX algorithm 89
SOUNDEX function 89-92, 96-97
SPEDIS function
COMPGED function and 92
COMPLEV function and 92
functionality 92-97
fuzzy matching and 81, 89-92
spelling distance between words 92-97
spread of data values, computing 244-247
spreadsheets, importing data from 196
SQRT function 266, 275-276
square roots 275-276
state codes
adding to addresses 345
converting from FIPS codes 337-338
converting from ZIP codes 344
converting to FIPS codes 339,341
converting to state names 339-341
state names
converting from FIPS codes 336-338
converting from state codes 339-341
converting from ZIP codes 342-345
statistics
See descriptive statistics functions
STD function
background information 238
functionality 245-246
performing $t$-test 264-266
STFIPS function 339, 341
STNAME function 339-341
STNAMEL function 340-341
storage length
of character variables 3-5, 111, 395-396
of numeric variables 395-396
strings
changing case of characters in 5-10
comparing 78-97
computing length of 111-114
concatenating 10,57-69
converting multiple blanks to single in 11-12
counting letters in 114-121
counting number of digits in 15-16
counting substrings in 114-121
counting words in 105-106
dividing into words $97-106$
extracting parts of 46-57
extracting phone numbers from 159
finding digits in random positions in 165-166
finding first numbers in 26
joining multiple 10,57-69
miscellaneous functions 121-142
removing blanks for 70-78
removing characters from 11-16
searching for characters in 16-46
searching for phone numbers in 153-155
substituting letters or words in 106111
substituting one for another 168-170
unpacking 49-50
STRIP function
ANY functions and 17
concatenation operator and 57
functionality $70,73,77-78$
NOT functions and 17
TRIMN function and 77
subjects, assigning randomly 294-302
substituting
letters or words in strings 106-111
one string for another 168-170
SUBSTR function
ARRAY statement and 49
CALL PRXPOSN routine and 162
CHAR function and 54

DO loops and 49
enciphering text using keys 417-418
encrypting and decrypting macros 419-420
extracting phone numbers from strings 159
finding digits in random positions in strings 165
FIRST function and 56
functionality 5, 25-26, 46-50
LENGTH function and 48,51
LENGTH statement and 51
locating ZIP codes in lists 158
SUBSTRN function and 52,54, 156
substrings, counting in strings 114-121
SUBSTRN function
functionality 52-54
LENGTH function and 53-54
LENGTH statement and 53-54
locating ZIP codes in lists 158
PRXSUBSTR function and 52
SUBSTR function and 52,54, 156
SUM function 240, 242-244
sums, computing 241-244
SURVEYSELECT procedure 291-292
SYMGET function 364, 368-369
\&SYSDAY automatic macro variable 366

## T

t distribution 312
$t$-test 264-266, 304-307
tab-delimited files, reading 101-102
table look-ups 328
TABULATE procedure 306
TAN function 353-354
telephone numbers
See phone numbers
test data sets, creating 372-373
test scores, averaging 253-255
text, enciphering and deciphering 416-418
text matches, locating lines with 145-147
text patterns, locating 152-167
TIME function 180, 185-187
time values
creating 180-188
defined 180
extracting parts of 192-195
working with intervals 197-211
TIMEPART function 195-197
TODAY function $180,184-185$
toggle switch, MOD function as 272-273
TRANSLATE function 106-109
TRANWRD function $106,109-111,168$
trigonometric functions
background information 351
basic 351-354
inverse 355-358
RETAIN statement and 354
TRIM function
ANY functions and 17
concatenation operator and $57,61,75$
functionality 73-75
NOT functions and 17
TRIMN function and 73,75-76
VERIFY function and 44
TRIMN function
functionality 75-76
STRIP function and 77
TRIM function and 73, 75-76
TRUNC function 225, 234-236
truncation functions
background information 225
returning numeric values 234-236
rounding and truncating numerical values 226-234
TTEST procedure 264, 306

## U

uniform distribution 290-302, 312
UNIFORM function
See RANUNI function
unpacking strings 49-50
unstructured data, reading 162-163
UPCASE function
COMPARE function and 80
functionality 6-7, 221

## 444 Index

UPCASE function (continued)
TRANSLATE function and 108
\$UPCASE informat 8
uppercase conversions 5-10
UTM coordinate system 349

## V

VALIDVARNAME system option
COMPGED function and 84
NLITERAL function and 136-137
NVALID function and 138-141
VARFMT function
CONTENTS procedure and 381
determining variable attributes 384
displaying data set attributes 388
functionality 381-382
variable information functions
background information 391-392
determining format information 403-405
determining informat information 406-410
determining information 392-403
OPEN function and 391
variables
See character variables
See macro variables
See numeric variables
VARLABEL function
CONTENTS procedure and 382
determining variable attributes 384
displaying data set attributes 388
functionality 382-383
VARLEN function
CONTENTS procedure and 383
determining variable attributes 384
displaying data set attributes 388
functionality 383
VARNAME function
averaging test scores 254
determining variable name 386
displaying data set attributes 388
functionality 384-385

VARNUM function
determining variable attributes 384
determining variable name 386
functionality 385
VARFMT function and 381
VARLABEL function and 382
VARLEN function and 383
VARNUM option, CONTENTS procedure 4
VARTYPE function 386-388
VERIFY function $44-46$
VFORMAT function 404, 408
VFORMATD function 404, 408-409
VFORMATN function 405,408
VFORMATNX function 403
VFORMATW function 405,408
VFORMATX function 403
VINFORMAT function 406, 409
VINFORMATD function 407, 409
VINFORMATN function 407, 409
VINFORMATW function 408
VLABEL function 399, 409
VLABELX function 399
VLENGTH function demonstrating 395-396 functionality 394
LENGTH function and 396
VNAME function 391-392, 396-399
VTYPE function 391-392, 394
VTYPEX function 392, 393-394
VVALUE function 327, 402-403

## W

\$w. informat 71
WEEK function 190-192
WEEKDAY function 191-192, 214
WHERE clause 91, 96, 377-378
WHICH format 327
wild cards
FORMAT statement and 224
regular expressions and 147-149
words
changing case of 5-10
converting to abbreviations $110-111$
counting in string 105-106
defined 97
dividing strings into $97-106$
searching for 40-41
spelling distance between 92-97
substituting in strings 106-111
writing, backwards 135-136

## X

XOR function 411
Y
Yates' correction, chi-square with 274
YEAR function 188, 191-192
YEARCUTOFF system option 217
YRDIF function
computing exact age $186,211,228$, 230
demonstrating 210-211
functionality 209-210
intervals supported 210

## Z

ZIP codes
computing distances between 335, 347-348
converting to FIPS codes 342
converting to state codes 344
converting to state names 342-345
locating in address list 157-158
ZIPCITYDISTANCE function 335, 347-350
ZIPFIPS function 342, 344
ZIPNAME function 342-344
ZIPNAMEL function 343-344
ZIPSTATE function 344-346

## Symbols

\& (ampersand) 346, 368

* (asterisk) 219, 221
: (colon) 44, 224
$=$ : comparison operator 80
|| (concatenation operator)
CALL routines and 57-58, 61
CAT function and 62
TRIM function and 75
- (dash) 13-14, 44
\$ (dollar sign) 404, 406
/ (forward slash)
as delimiters 145,150
in date formats 44
in regular expressions 145, 147
() (parentheses)
array elements and 219
regular expressions and 147
removing from phone numbers 13-14
\# (pound sign) 104
?? modifier 323



# Gain Greater Insight into Your SAS ${ }^{\circledR}$ Software with SAS Books. 

Discover all that you need on your journey to knowledge and empowerment.


