The How To Pass on Your First Try Certification Study Guide

SAS

Certified Base Programmer for SAS 9 Certification Exam Preparation

Course in a Book for Passing the SAS Certified Base Programmer for SAS 9 Exam

William Manning

1 Foreword

This Exam Preparation book is intended for those preparing for the SAS Certified Base Programming for SAS 9 Exam.

The Art of Service is an Accredited Training Organization and has been training IT professionals since 1998. The strategies and content in this book is a result of experience and understanding of the SAS programming, and the exam requirements.

This book is not a replacement for completing the course. This is a study aid to assist those who have completed an accredited course and preparing for the exam.

Do not underestimate the value of your own notes and study aids. The more you have, the more prepared you will be.

While it is not possible to pre-empt every question that may be asked in the SAS Certified Base Programmer exam, this book covers the main concepts covered within the SAS Programming discipline.

The SAS Certified Base Programmer for SAS 9 is the ideal certification for those relatively new to SAS programming or new to SAS certification. It is also the principle certification for many of the advanced certifications available from SAS.

These study notes and sample exam questions in this book will allow you to more easily prepare for an SAS Programming exam.

Ivanka Menken
Executive Director
The Art of Service

2 Table of Contents

3 SAS Certified Base Programmer for SAS 9

SAS Certified Base Programmer for SAS 9 covers a basic programming knowledge to develop efficient code to meet business requirements. The certification is geared towards individuals who are new to SAS programming or new to SAS certifications. The certification program for Base Programmer for SAS 9 is the basis for all other certifications provided by SAS.

A SAS Base Programmer should have some experience in data management and SAS programming, specifically with tasks related to:

- manipulating and transforming data
- importing and exporting raw data files
- combining SAS date sets
- creating basic reports
- correcting errors in data syntax and programming logic.

Certification is valid for three years, at which time a recertification exam must be completed.

4 Exam Specifics

Exams are delivered in a secure environment, proctored, and timed by Prometric. Details about the exam include:

- 70 multiple-choice questions
- Score of 65%
- 2 hour testing time.

Tests can be scheduled online. Prometric requires a 24-hour waiting period for first-time registrations of new candidate. The first-time register will receive an email confirmation.

If a candidate fails the exam, they must wait a minimum of 30 calendar days before attempting again.

Candidates are required to bring two forms of identification to the testing center on the day of their test. They are recommended to arrive 15minutes before the test. If they are more than fifteen minutes late, a seat may not be available, nor a refund guaranteed.

5 Exam Prerequisites

There are no prerequisites for SAS Certified Base Programmer for SAS 9.

6 SAS Fundamentals

Key components of SAS

- Programs
- Libraries
- Files
- Data sets.

6.1 SAS Programs

SAS programs are used to access, manage, analyze, or present data. Most SAP programs are a combination of DATA and PROC (procedure) steps.

DATA steps create or modify data sets, specifically:

- Entering data into a SAS data set
- Checking for errors in the data
- Computation of data
- Producing new SAS data sets
- Producing customer-designed reports.

PROC steps are pre-defined procedures to analyze and process data in a SAS data set.

PROC steps are used to manipulate the data typically for reporting purposes. As a result, PROC steps can:

- Create new SAS data sets
- List, sort, and provide summaries of data
- Produce statistical results
- Produce plots and charts.

6.1.1 Formatting and Behavior of SAS Programs

- Programs consist of statements.
- Statements start with a keyword.
- Statements end with a semicolon.

Statements are in free format:

- Can begin and end anywhere on a line
- Can exist on multiple lines
- Can exist with multiple statements on a single line

Statements can be in uppercase or lowercase.
Case sensitivity is typically resolved using quotation marks.
Separation of words is resolved using spaces or special characters.

6.1.2 Processing Programs

Upon submission of a SAS program, SAS will read the statements. While statements are being read, error checking is in place.

New steps begin with a DATA and PROC statement. The completion of a step is identified by a RUN or QUIT statement.

SAS will read the step completely. When a new step is found, SAS stops reading and will execute the step it just read.

When a step is executed, the processing activity is logged with the results of the processing. The SAS log is a collection of these messages as well as any errors that occurred.

General results of processing:

- Creation of a SAS data set
- Production of SAS log messages
- Creation of a report
- Sorting and managing data.

6.2 SAS Libraries

Libraries are collections of files. Implementation of libraries depends on the operating environment. A library can be a physical or logical collection.

Summary of libraries in different operating environments:

- Windows – stored in the same directory
- UNIX – stored in the same directory
- OpenVMS – stored in the same directory
- CMS – same file type
- z/OS – specially formatted host data

Files can be stored temporarily or permanent. When a library name is ‘Work’ or unspecified, files are stored in a temporary SAS data library. Temporary files are deleted at the end of a session.

Files are permanently stored in libraries that have a specified name other than ‘Work’.

6.2.1 File Referencing

To reference, a two-level name is used. The format of a two-level name is:

libref.filename.

libref is ‘library reference’, the SAS data library containing the file.
filename is the name of the specified file.

If there is no libref identified or ‘work’, the file reference is a temporary file. All other files are permanent.

Several libraries are defined by default:

- ‘Sashelp’ is a permanent read-only library containing sample data and other files that control the behavior of the SAS
- ‘Sasuser’ is a permanent library containing SAS files that

stores personal settings in the Profile catalog
- ‘Work’ is a temporary library for files.

Defining libraries is often the first step in setting up a SAS session.

Assign a library name (libref):
- Use the LIBNAME statement
- Format:
  LIBNAME *libref 'SAS-data-library'*
- *libref* is 1 to 8 characters long beginning with a letter or underscore and containing only letters, numbers, or underscores
- *SAS-data-library* is the name of the library where files are stored
- Multiple LIBNAME statements can be used to assign as many librefs as required.
- LIBNAME statements are global and stay in effect until the libref is modified, canceled or the SAS session ends.

The files in the library are inaccessible if the library is deleted or the session ends. The files still exist on the operating system.

Comparison of physical names of SAS data within operating systems for the library “sample”:
- Windows – c:\sample\data
- UNIX - /users/month/sample/sasdata
- OpenVMS – dua0:[month.fitness]
- CMS – b
- z\OS – month.sample.sasdata

6.2.2 Referencing Other Formats

The LIBNAME statement can be used to reference files created by other software programs.

To read or write to foreign files, an engine for the file type must be used. A SAS engine is a set of internal instructions used by SAS. Sometimes, the appropriate engine is brought up automatically; other times, the engine must be specified.

To specify the engine to be used

- Use the LIBNAME statement
- Format:
  LIBNAME *libref engine 'SAS-data-library'*
- *libref* is 1 to 8 characters long beginning with a letter or underscore and containing only letters, numbers, or underscores
- *engine* is the name of the library engine used in the operating system
- *SAS-data-library* is the name of the library where files are stored

Interface library engines support read-only access to BMDP, ORISIS, and SPSS files. Physical filenames associated with a libref are actual filenames (a rule exception).

- Example: Using libref = Differ and the SPSS engine for the c:\newdata.dat in the windows operating environment: libname differ spss 'c:\newdata.dat'

Other types of data that can be read or written to are dependent on the operating environment and the SAS/ACCESS products that are licensed, including:

- Relational Databases
  ORACLE
  SYBASE
  Infomix
  DB2 for z/OS (OS/390)
  DB2 for UNIX and PC
  Oracle Rdb
  ODBC
  CA-Openingres

- Non-relational Files
  ADABAS
  IMS/DL-I
  CA-IDMS
  SYSTEM 2000

- PC Files
  Excel (.xls)
  Lotus (.wkn)
  DBF
  DIF

6.2.3 View Library Contents

The CONTENTS procedure creates output to:

- describe the contents of a library
- describe the descriptor information for a SAS data set.

To create SAS output of library contents:

- Use the PROC CONTENTS statement
- Format:
  PROC CONTENTS DATA=libref._ALL_NODETAILS;
  RUN;
- *libref* is the libref assigned to the SAS library
- _ALL _ requests a listing of all files in the library
- NODETAILS prevents detailed information about each file from being in the output.

PROC DATASETS can be used to view the contents of a SAS library or SAS data set with a CONTENTS statement.

The difference between the two procedures is the type of statement and the defaults for *libref*:

- PROC DATASETS is an interactive procedure requiring a QUIT statement
- The PROC DATASETS default is DATA=option
- The PROC CONTENTS default is either Work or User.

To create SAS output of library contents:

- Use the PROC DATASETS statement with CONTENTS
- Format:
  PROC DATASETS;
     CONTENTS DATA=libref._ALL_NODETAILS;
  QUIT;
- CONTENTS describes the contents of SAS data sets

specified and prints the directory of the library
- *libref* is the libref assigned to the SAS library
- _ALL _ requests a listing of all files in the library
- NODETAILS prevents detailed information about each file from being in the output.

When viewing descriptor information for a specific data set, the variables are listed alphabetically by default.
Descriptor information can be viewed in logical position, or creation order, by using the VARNUM option:

- PROC DATASETS;
  CONTENTS DATA=libref VARNUM;
  QUIT;

6.3 SAS Data Sets

SAS data sets are one type of SAS file.
To process data, it must be in the form of a data set.
Processing can consist of:

- Accessing data
- Analyzing, managing, and presenting data.

Data sets must have a name. Valid data set names can:

- Be 1 to 32 characters long
- Begin with a letter, uppercase or lowercase, or an underscore
- Consist of any combination of letters, numbers, and underscores.

Data set has two parts: a descriptive portion and a data portion that the data set can locate.

6.3.1 Descriptive Portion

Contains information about the data set, including:

- data set name
- data type
- creation date and time
- number of observations
- number of variables
- number of indexes.

Contains information about the variables in the data set, including:

- name
- type
- length
- format
- informat
- label.

6.3.2 Data Portion

Collection of data values arranged in a table. Observations refer to the rows in the table. Each row is a collection of data values that relate to a single object.

Variables refer to the columns in the table. Each column is a collection of values that describe a particular characteristic. Each variable in a data set has a series of attributes. Variables of the same name can aid in combining SAS data sets. To combine data sets, variables must be the same type.

A variable name conforms to SAS naming conventions.
Variable names follow the same rules as data set names:

- Be 1 to 32 characters long
- Begin with a letter, uppercase or lowercase, or an underscore
- Consist of any combination of letters, numbers, and underscores.

Variable types are either character or numeric:

- Character variables can contain any value
- Numeric variables can contain only numeric values
- Representation of missing values is determined by variable type:
    - A blank represents a missing value for character values
    - A period represents a missing value for numeric values.

Variable length identifies the number of bytes used to store the variable. Length is dependent on type:

- Character variables can be up to 32,767 bytes long
- Numeric variables have a constant default length of 8 with an infinite number of digits possible
- Numeric variables have a constant length because they are stored as floating-point numbers
- A different length can be specified for numeric variables.

Variable format affects the way data values are written.

- Formats are provided by SAS software or created by a programmer
- The appropriate format must be selected to write value in a given formatting. The maximum width of the written value usually required to be specified
- Some formats require specification of the number of decimal places
- A format can be permanently assigned to a variable in a SAS data set or temporarily assigned in a PROC step to control output appearance.

When data values must be read into a SAS data set, how they are read must be specified. Reading specifications are controlled by informats.

Variables can have labels:

- Labels are descriptive text up to 256 characters long
- By default, variables are identified by their names: for more information on the variables, labels are applied.

6.4 SAS System Outputs

Result formats for SAS formats can be specified:

- Traditional SAS listing
- HTML document
- Both a listing and HTML document.

The appearance of the output can be controlled, specifically (for SAS listings):

- line size (the maximum width of the log and output)
- page size (the number of lines per printed page)
- page numbers displayed
- data and time displayed.

To modify system options, use an OPTIONS statement (a global statement).

- OPTIONS:
    - options identifies one or more system options to be changed
    - The system options available are dependent on the operating system.

Survey of common options:

NUMBER | NONUMBER	Page numbers appear by default, so NONUMBER prevents the printing of page numbers.
DATE | NODATE	Date and time appear by default, so NODATE prevents the printing of date and time.
PAGENO=option	Specifies the beginning page number for report. The default is 'page 1'.
PAGESIZE=option	Specifies how many lines on each page of the output.
LINESIZE=option	Specifies the width of the print line. Observations that are unable to fit on a single line will continue on a different line.
TEARCUTOFF=option	Fulfills Year 2000 Compliance.

OBS=	Specifies the last observation to be processed. The default is MAX.
FIRSTOBS=	Specifies the first observation to be processed. The default is 1.
FIRSTOBS = and OBS=	Specifies a range of observation to be processed.
FORMCHAR= 'formatting-character'	Specifies the formatting characters for the output device.
FORMDLIM= 'delimiting-character'	Specifies the character used to delimit page breaks. Default is null.
LABEL | NOLABEL	Can replace variable names with descriptive labels. The default is LABEL.
SOURCE | NOSOURCE	Controls if SAS source statements are written to the log. Default is SOURCE.

FIRSTOBS= and OBS= can be used as data set options instead of system options.
A data set option is set in parenthesis following the input data set name.

- Format:
  ```
  options firstobs=8 obs=16;
  proc print data=newdata.set (firstobs=5 obs=22);
  run;
  ```

To view the current setting of one or all SAS system options, use the OPTIONS procedure.

- Format:
  PROC OPTIONS <*option(s)*>*;*
  RUN;

7 *Creating SAS Data Sets*

7.1 Raw Data Files

A raw data file is an external text file. Raw data files are non-proprietary. Many software programs use raw data files.

The external file contains data values which are organized in fields. Fields may be difficult to locate in raw data format. Fixed fields ensures the values for specific variables are located in the same position for all records.

7.1.1 Creating a Data Set with Fixed Fields

1. The SAS library used to store the data set must be referenced.
2. A DATA step program is written to read the raw data file and create a SAS data set
   - The following instructions must be provided to SAS by the DATA step in order to read the raw data file:
     - Name or location of the external file
     - Name for the new SAS data file
     - Reference for the external file
     - Description of the data values to be read
   - Additional SAS statements can be used to customize the data.
3. A PROC PRINT step can produce a list report to display the data values.

Basic statements for construction:

LIBNAME statement	Reference a SAS data library.
FILENAME statement	Reference an external file.
DATA statement	Name a SAS data set.
INFILE statement	Identify an external file.
INPUT statement	Describe data.

RUN statement	Execute DATA step.
PROC PRINT statement	List data.
RUN statement	Execute final program statement.

7.1.2 Creating References

To reference a permanent SAS library, a LIBNAME statement is required:

- Use the LIBNAME statement
- Format:
  LIBNAME *libref 'SAS-data-library'*
  - *libref* is 1 to 8 characters long beginning with a letter or underscore and containing only letters, numbers, or underscores
  - *SAS-data-library* is the name of the library where files are stored.

A LIBNAME statement is not required for all situations, such as temporary SAS data set or if the libref has been automatically assigned for the permanent library by SAS.

The external file containing the raw data must be referenced using a FILENAME statement:

- Use the FILENAME statement
- Format:
  FILENAME *fileref 'filename'*
  - *fileref* is a name associated to an external file and must be 1 to 8 characters long beginning with a letter or underscore and containing only letters, numbers, or underscores
  - *filename* is the fully qualified name or location of the file.

When a fileref is associated with an individual external file, the fileref is specified in subsequent SAS statements and commands.

When a fileref is associated with an aggregated storage location, the fileref is specified with the individual filename in parenthesis.

When reading raw data, the INFILE statement identifies which file is storing the data.
To identify the external file where data is stored:

- Use the INFILE statement
- Format:
  INFILE *file-specification <options>;*
    - *file-specification* takes the form of fileref from a previously defined file reference or 'filename' to point to the actual name and location of the file
    - *options* describes the characteristics of the input file and how the raw data should be read.

LIBNAME and FILENAME are global statements which are in place until they are changed, canceled, or the SAS session ends.

7.2 The Data Step

7.2.1 Writing a Basic Step Program

The start of a DATA step is the DATA statement. Within the DATA statement, the SAS data set to be created is named.

To begin a DATA step and name a SAS data set:

- Use the DATA statement
- Format:
  DATA *SAS-data-set-1 (...SAS-data-set-n>;*
    - *SAS-data-set-n* is the name of the data set(s) to be created

Columns are the common input style specifying actual locations for values. Columns should not be used in all situations, but only when data is:

- consists of standard character and numeric values
- in fixed fields.

Standard numeric data values contain only:

- numbers
- decimal points
- scientific notations (e-notation)
- positive or negative numbers.

Nonstandard numeric data values include:

- contain special characters such as % or $
- date and time values
- fractions
- integer binaries
- real binaries
- hexadecimal numbers.

Raw data organized in columns, or fixed format, allow a beginning and ending column for each file to be specified. Raw data that is not organized in columns is considered in free format. Free format cannot be read by column input.

7.2.2 Describing Data

The INPUT statement is used to describe the fields of raw data. For each field of raw data to be read, the information required for the INPUT statement includes:

- a valid SAS variable name
- a variable type (character or numeric)
- a range identifying starting and ending columns.

To describe the field of raw data to be read and placed in the SAS data step:

- Use the INPUT statement
- Format:
  INPUT *variable; <$> startcol-endcol...*
    - *variable* is the SAS name that you assign to the field
    - The dollar sign indicates the variable type as character. To indicate numeric, no dollar sign should exist
    - *startcol* is the starting column for the variable
    - *endcol* is the ending column for the variable.

Using column input, you can:

- read any or all fields from the file
- read the filed in any order
- specify only starting columns when values occupy only one column.

Variable names conform to SAS naming conventions:

- Be 1 to 32 characters long
- Begin with a letter, uppercase or lowercase, or an underscore
- Consist of any combination of letters, numbers, and underscores.

To verify data, use the OBS=option in the infile statement. Using OBS=n allows a portion of the data to be processed.

The log will verify that the raw data file was read correctly, typically showing:

- The number of records read
- The number of observations and variables created.

If invalid data occurs, a note in the log will occur which include the variable that erred, a column ruler and actual data line.

To view the data processed, use a PROC PRINT step.

7.2.3 Creating and Modifying Variables

To modify existing values or create new variables, an assignment statement can be used in any DATA step.

To modify or create variables:

- **Format:**
  variable=expression;
  - *variable* is the name of the new or existing variable
  - *expression* is any valid SAS expression.

SAS expressions are used to:

- transform variables
- create new variables
- conditionally process variables
- calculate new values
- assign new values.

Expressions are instructions that consist of a sequence of operands and operators:

- Operands are variable names or constants in character or numeric format
- Operators are special-characters, grouping parenthesis, or functions.

Arithmetic operators allow calculations to be performed. Arithmetic operators include:

negative prefix	(-)
exponentiation	(**)
multiplication	(*)
division	(/)
addition	(+)
subtraction	(-)

Arithmetic operators have priorities associated with them to assist in properly calculating an expression when more than one operator exists in the expression. To use priorities properly:

- Higher priorities are performed before lower priorities
- If consecutive operators have the same priority to be performed:
    - from right to left within priority I
    - from left to right within priorities II and III
- The control of operations can be done with parentheses.

Comparison operators express a condition. Comparison operators include:

- equal to (= or eq)
- not equal to (^= or ne)
- greater than (> or gt)
- less than (< or lt)
- greater than or equal to (>= or ge)
- less than or equal to (<= or le)
- equal to one of a list (in)

Logical operators create a link between sequences of expressions. Logical operators (boolean) include:

- both and (AND or &)
- either or (OR or |)
- not (^)

Data values can be assigned to variables using date constants. Constants are represented by 'ddmmmyy' or ddmmmyyyy' followed by a D

- Format:
  'ddmmmyyyy' D

Some situations require processing to be done only when an observation meets a specific condition, which uses a subsetting IF statement to indicate the condition. To express a condition:

- Use the IF statement
- Format:
  IF *expression;*
  - expression is any valid SAS expression:
  - If the expression is true the DATA step continues
  - If the expression is false, no further statements are processed.

7.2.4 Using Instream Data Lines

The INFILE statement identifies an external file to read. The DATALINES statement identifies data in the SAS program to be read, and is the last statement in the DATA step.

To read instream data lines:

- Use the DATALINES statement
- Format:
  DATALINES;
  Only one DATALINES statement for each DATA step

- CARDS is an alias for DATALINES and can be used as well
- If the data contains semicolons, use DATALINES4;;;;

7.2.5 Creating a Raw Data File

To create a raw data file:

- Use the SET statement
- Format:
  DATA _null_;
    SET dataset;
  _null _ allows the DATA step to be used to without creating a SAS data set
  SET identifies the SAS data set to read

Writing observations from the data set into the raw data can be done using FILE and PUT statements.

- The FILE statement specifies the output file.
- The PUT statement how the lines should be written to the file.

To specify an output file:

- Use the FILE statement
- Format:
  FILE *file-specification;*
  - *file-specification* takes the form of fileref from a previously defined file reference or 'filename' to point to the actual name and location of the file
  - *options* describes the characteristics of the input file and how the raw data should be read.

To describe the lines written to the raw data:

- Use the PUT statement
- Format:
  PUT *variable startcol-endcol...;*
  - *variable* is the name of the variable being written
  - *startcol* identifies the line to begin writing the value
  - *endcol* identifies the line to end writing the value.

7.3 Processing DATA Steps

When a DATA step is submitted, the SAS processes the DATA step and creates a new SAS data set. The process consists of two phases:

- The compilation phase scans each statement for errors. An error will prevent further processing. When complete with no errors, the descriptor portion of the data set is created
- The execution phase, the DATA step reads and processes the input data. By default, the DATA step is executed once for each record in the input file.

7.3.1 Compiling the DATA Step

To start the compilation phase, the input buffer is created:

- The input buffer is an area of memory used to hold a record from the external file
- It is only required when reading raw data
- Refers to logical, not physical, memory.

The creation of a program data vector begins after the input buffer is created. The program data vector is an area of memory where the SAS will build the data set one observation at a time:

- Refers to logical, not physical, memory
- Contains two variables automatically which are used for processing but not written to the data set
- _N_ is the number of times the DATA step begins execution
- _ERROR_ is the occurrence of an error caused by data. The default is 0, identifying no errors. One of more errors existing, the value is 1.

Each statement is scanned for syntax errors, which include:

- missing or misspelled words
- invalid variable names
- missing or invalid punctuation
- invalid options.

INPUT statements add a slot to the program data vector for each variable in the new data set.
Variable attributes are identified when the variable is first encountered.

At the end of the DATA step, the compilation phase is complete.
The descriptor portion of the new data set is created, which includes:

- the name of the data set
- the number of observations and variables
- the name and attributes of the variables.

7.3.2 Executing the DATA Step

The execution phase will create the data portion of the data set which contains the data values. The input raw data file is read from the program data vector. Observations are new data that are written out. One observation is created for each record being inputted.

At the start of the execution phase, the file is checked for errors.

- _N_ is set at value 1.

The INFILE statement is read to identify the location of the raw data. Any INPUT statements are read from the input buffer:

- an input pointer will track of the reading position
- the input statement starts at column 1 by default.

At the end of the DATA step:

- Values in the program data vector are written as the first observation
- The value of _N_ is incremented by 1 and a return to the top of the DATA step
- Variable values in the program data vector are re-set to missing.

The DATA step becomes a loop with each pass through called an iteration.

The execution phase continues until the end-of-file marker in the raw data file is reached.

7.4 Debugging DATA Steps

7.4.1 Errors During Compilation

During the compilation phase, many errors can be detected, including:

- misspelled keywords and data set names
- missing semicolons
- unbalanced quotation marks
- invalid options.

The SAS will interpret some syntax errors. If the error is not interpreted:

- prints the word ERROR in the log followed by the error message
- compiles the step, but does not execute
- prints a warning - NOTE: The SAS System stopped this step because of errors.

7.4.2 Errors During Execution

Errors can occur during the Execution phase. How errors are dealt with is dependent on the type of error. Actions taken can include:

- A note warning, or error message in the log
- Values stored in the program data vector displayed in the log
- Processing continues
- Process stops.

7.4.3 Performing Testing

DATA steps can be compiled and executed without creating a dataset, by using a NULL keyword. Used to detect common error messages and saves on development time.

To create a NULL Data Set:

- Use the DATA statement
- Format:
  DATA _null_;

All compilation and execution errors are written to the log, without creating a data set. After correcting errors, the NULL statement can be replaced with the appropriate data set name.

The OBS= option in the INFILE statement can limit the number of observations that are read or created during execution.

The PUT statement can be used to examine variable values. A personal message can be created in the log.

IF-THEN/ELSE statements can be used to conditionally test for values.

7.4.4 Creating Message Strings

Several forms of messages can be added to the log using the PUT statement. To specify a character string, enclose the string with quotations marks:

- Format:
  PUT 'NOTE: Character Strings' ;

To specify one or more data set variables for a single iteration of the DATA step, add the expression 'code type':

- Format:
  PUT 'NOTE: Data Set Variables:'
  code type;
    - This expression will display only the value of the variable.
- Format:
  PUT 'NOTE: Data Set Variables:'
  code= type=;
    - This expression will display the name and value of the variable.

To specify the automatic variables, _N_ and _ERROR_:

- Format:
  PUT 'NOTE: Automatic variables:'
  code= _n_= _error_=;

To specify all variables and automatic variables, use the _all_ expression:

- Format:
  PUT 'NOTE: All variables:' _all_;

Conditional processing can be used to flag errors or out of range data, using IF-THEN/ELSE statements:

- Format:
  IF *desired option* THEN
  desired action
  ELSE PUT 'NOTE: Conditional processing';

7.5 SAS Data Sets

SAS Data Sets are created from either raw data sets or other SAS data sets.

7.5.1 Reading SAS Data Sets

To create a data set, the libraries have to be referenced first:

- The library where the data is stored
- The library where the data will be stored

Then DATA and SET statements are used to create a data set using values from another data set.

To create a data set from values found in another data set:

- Use the **DATA** statement and **SET** statement
- **Format:**
  DATA *SAS-data-set*
  SET *SAS-data-set*
  RUN;
  - *SAS-data-set* in the DATA statement is the libref. filename of the SAS data set to be created
  - *SAS-data-set* in the SET statement is the libref. filename of the SAS data set to be read

Sometimes, the data to be used needs some manipulation. Some common actions include:

IF *variable condition* THEN *action*;
IF *variable limit;*
	Use a subset of data.
	Variable conditions can be <, >, <=, >=,
	Variable limits are typically equalities or inequalities.

DROP *variable1, variable2...;*	Drop any unwanted variables.
NewVariable = *variable condition1*;	Create or modify a variable.
LENGTH variable;	Specify a variable's length.

IF *variable condition* THEN *action*; ELSE *alternate action;*	Conditional statements.
LABEL variable;	Label a variable.
FORMAT variable;	Format a variable.

DROP= and KEEP+ statements are options that can be used whenever data sets are named, namely either the DATA or SET statements:

- If a variable should be processed but not appear in the new data set, use the DROP= option in the DATA statement
- If a variable should not be processed nor appear in the new data set, use the DROP= option in the SET statement.

A BY statement can be used to group observations:

- Datasets can be sorted by the value of one or more variables specified from a BY statement
- Two temporary variables can be created: FIRST.variable and LAST.variable. The value for both is either 1 or 0 and they identify the first and last observation in each BY group.

Multiple BY variables can be used to assign conditions to data sets. The result of multiple BY variables:

- FIRST.variable for each variable is set to 1 for the first occurrence of a new value
- A change in the value of a primary BY variable forces LAST.variable to be 1 for secondary BY variables.

Observations in the input data set are read as they appear in the physical file, or sequentially.
Sequential reading can be bypassed using a POINT= option.

To read observations directly:

- Use the POINT= statement
- Format:
  POINT=*variable*;
  - *variable* specifies a temporary numeric variable containing the observation number of the desired observation and must be given a value before executing a SET statement.

Reading observations continues until an end-of-file marker is reached: the use of a POINT= option will force the observation reading to never reach the end-of-file marker.
The condition causes continuous looping; to resolve:

- Use a STOP statement
- Use programming logic that checks for an invalid value of the POINT= variable

7.5.2 Writing Observations

To override the way the DATA step writes observations to output, an OUTPUT statement can be used. Observations are added to the data set only when the OUTPUT statement is executed.

To control the observations written to a data set:

- Use the OUTPUT statement
- Format:
  DATA *SAS-data-set;*
  SET *SAS-data-set;*
  OUTPUT <*SAS-data-set(s)*>;
  STOP;
  - *SAS-data-set*(s) names the data set(s) where observations will be written
  - If a data set is not named, the OUTPUT statement impacts all data sets named by the DATA step.

In some cases, identifying the end of the data set is required to control the observations that are written, such as identifying totals. To identify the last observation in a DATA set:

- Use the END= option
- Format:
  END= *variable*
  - *variable* is the creation of a temporary variable containing an end-of-file marker. The variable is initialized at 0 and set to 1 when the SET statement reads the last observation of the data set.

The variable specified in the END= option will not be added to the data set. END= and POINT= cannot be used in conjunction.

7.5.3 Running a Data Step

Reading SAS data sets is similar to reading raw data, except that SAS retains the values of the variables from each observation. Includes a compilation and execution phase.

Compilation steps:

- Program data vector is created
- Each statement is scanned for syntax errors
- Any created variables are added to the program data vector
- At the bottom of the DATA step, compilation ends and the descriptor portion of the new SAS data set is created.

Execution steps:

- DATA step executes for each observation in the input data set
- The value of _N_ is 1 and the value of _ERROR_ is 0; all other variables are initialized to missing
- The SET statement reads the first observation from the input data set and write the values to the program data vector
- Any assignment statements are executed
- At the end of the first iteration of the DATA step, the

values in the program data vector are written to the new data set
- The value of _N_ is set to N+1
- The values of variables read from the SAS data set with the SET statement are retained and all other variables set to missing
- The process continues until all observations are read.

7.6 Combining SAS Data Sets

Methods of combining data sets:

- One-to-one reading – will create observations that contain all variables from each contributing data set based on relative position in each data set
- Concatenating – Appends observations from one data set to another
- Interweaving – Intersperses observations from two or more data sets, based on one or more common variables
- Match-merging – Matches observations from two or more data sets into a single observation in a new data set based on the values of a common variable.

7.6.1 One-to-One Reading

To create a data set combining multiple data sets using one-to-one:

- Use the DATA statement and multiple SET statements
- Format:
  DATA *output-SAS-data-set;*
  SET *SAS-data-set-1;*
  SET *SAS-data-set-n;*
  RUN;
    - *output-SAS-data-set* names the data set to be created
    - *SAS-data-set-n* names the data sets to be read.

The criteria used by one-to-one readings to select data:

- The new data set contains all variables from all input data sets
- If data sets have variables of the same name, the values from the last data set overwrite the values read from previous data sets
- The number of observations in the new data set will be the same as the number of observations in the smallest original data set
- Observations are combined based on their relative position, that is, the first observations from each data set are combined, and so on.

7.6.2 Concatenating

To create a data set combining multiple data sets using concatenation:

- Use the DATA statement and the SET statement
- Format:
  DATA *output-SAS-data-set;*
  SET *SAS-data-set-1 SAS-data-set-n;*
  RUN;
    - *output-SAS-data-set* names the data set to be created
    - *SAS-data-set-n* names the data sets to be read.

The criteria used by Concatenation readings to select data:

- The new data set contains all variables and observations from all input data sets
- All observations are read from the data sets listed in the SET statement
- Conditions to watch for:
    - Instances of a common variable must have the same type attribute, or processing is stopped and an error message issued
    - Common variables with different attributes other than type will retain the attribute from the first data set containing the variable, including length, label, format, and informat.

7.6.3 Interweaving

To create a data set combining multiple data sets using interweaving:

- Use the DATA statement and the SET statement
- Format:

  DATA *output-SAS-data-set;*
  SET *SAS-data-set-1 SAS-data-set-n;*
  BY *variable(s);*
  RUN;

  - *output-SAS-data-set* names the data set to be created
  - *SAS-data-set-n* names the data sets to be read
  - *variable(s)* specifies one or more variables used to interweave.

The criteria used by Interweaving readings to select data:

- Observations in each BY group in each data set is read sequentially in the order of the data sets and BY variables are listed
- The new data set contains all variables from all input data sets and contains the total number of observations from all input data sets
- Observations are grouped by a common variable.

7.6.4 Simple Match-Merging

To create a data set combining multiple data sets using match-merging:

- Use the DATA statement and the MERGE statement
- Format:

  DATA *output-SAS-data-set;*

  MERGE *SAS-data-set-1 SAS-data-set-n;*

  BY <DESCENDING> *variable(s);*

  RUN;
  - *output-SAS-data-set* names the data set to be created
  - *SAS-data-set-n* names the data sets to be read
  - variable(s) specifies one or more variables used to merge
  - DESCENDING identifies if the input data set should be sorted from largest to smallest. The absence of DESCENDING has the input data set sorted in ascending order.

The criteria used by Interweaving readings to select data:

- The new data set contains all observations from all input data sets, though options can be in place that would reduce the number of observations found in the output
- Each observation of each data set is checked sequentially for matching BY values and writes the combined observation to the new data set
- If an input data set doesn't have any observations for a particular value for a same-name variable, missing values will be displayed in the output.

7.7 Processing Match-Merge

7.7.1 Match-merge Compilation and Execution

To compile data sets used for a merge-match:

- Descriptor portions of all data sets listed in the MERGE statement are read
- The remaining DATA step program is read for variables and options impacting the merge
- A program data vector is created for the merged data set
- A tracking pointer is assigned for each data set listed in the MERGE statement.

After compilation, the observations are sequentially match-merged by moving the tracking pointers through each observation of each data set and checking for BY values that match:

- If a match is found, the observation is written to the program data vector in the order the data sets appear in the MERGE statement.
- The value of same-named variable retain the value of the last data set where the variable appears
- If a match is not found, the order of the values is determined by SAS and the observation is written to the program data vector.

By default, all observations are written to the program data vector, including observations with missing data and no matching BY values. Observations written to the PDV are written to the output data set.

If an observation contains missing values for a variable, the observation in the output data set contains missing values.
If a BY variable exists, observations that have missing values will appear at the top of the output data set.
If an input data set doesn't have a matching BY value, observations in the output data set contains missing values for the variables.

7.7.2 Renaming Variables

Same-named variables can appear in multiple data sets to be merged, which will overwrite each other based on the order the data set appears in the MERGE statement.
Sometimes, the name of a variable in one or more data set needs to be renamed to prevent being overwritten.

To rename variables:

- Use the RENAME-data set option and the MERGE statement
- Format:
  DATA *output-SAS-data-set;*
      MERGE *SAS-data-set-1*(RENAME=*old-variable-name*=*new-variable-name*))
      SAS-data-set-n;
      BY *variable(s);*
  RUN;
    - *output-SAS-data-set* names the data set to be created
    - *SAS-data-set-n* names the data sets to be read
    - variable(s) specifies one or more variables used to merge
    - RENAME=options follows the name of each data set that contains the variable(s) to be renamed
    - *old-variable-name* identifies the name of the variable to be renamed
    - *new-variable-name* identifies the new name for the variable.

Variables can be renamed for data sets specified by the DATA and SET statements.

7.7.3 Excluding Unmatched Observations

Match-merging combines all observations in all input data sets. Sometimes, the desired result displays only those observations that match for two or more specific input data sets.
Exclusion of unmatched observations can be performed using the IN=data set option and a subsetting IF statement in the DATA step:

- The IN= data set option creates and names the variable that identifies if a data set contributed data to the current observation
- The subsetting IF statement checks the IN= values and writes to the merged data set only those observations where IN= is specified.

To exclude unmatched observations:

- Use the IN= option and the MERGE statement
- Format:

```
DATA output-SAS-data-set;
    MERGE SAS-data-set-1(IN=variable1)
        SAS-data-set-n (IN=variablen) ;
    BY variable(s);
    IF variable1=1 and variablen=1
RUN;
```

  - *output-SAS-data-set* names the data set to be created
  - *SAS-data-set-n* names the data sets to be read
  - variable(s) specifies one or more variables used to merge
  - IN=option, in parentheses, follows the data set name
  - *variablen* names the variable to be created
  - new-variable-name identifies the new name for the variable
  - The value of the variable is 1 if the data set is contributing to the current observation and 0 if it is not
  - IF statement provides the condition that an observation is created only if both conditions are met.

Variables can be dropped or kept by using the DROP= and KEEP=data set options.

8 *Working with Data*

Raw data can be organized in several ways:

- arranged in columns, or fixed fields
- arranged without columns, or free format.

Raw data can contain:

- standard data without any special characters
- nonstandard data with special characters.

SAS has three input styles

- column input
- formatted input
- list input.

8.1 Column Input

Column input allows fields to be read in any order. Character variable values that contain embedded blanks can be read. Missing data does not require a placeholder.

A blank field will be read as missing and will not cause other fields to be misread. Fields or parts of a field can be re-read. Fields do not have to be separated by blanks or other delimiters.

8.1.1 Numeric Data

When raw data is organized in fixed fields:

- use column input to read standard data only
- use formatted input to read both standard and nonstandard data.

Standard numeric data values contain:

- numbers
- decimal points
- numbers in scientific notation
- positive and negative numbers.

Nonstandard numeric data values contain:

- special characters
- data and time values
- fractions
- integer binary
- real binary
- hexadecimal forms.

To read both standard and nonstandard data in fixed fields:

- Use an INPUT statement
- Format:
  INPUT *<pointer-control> variable informat.;*
    - *pointer-control* places the input pointer in the appropriate column
    - *variable* is the name of the variable to be created
    - *informat* is the special instruction specifying how SAS reads raw data.

Two column pointer controls are utilized:

- the @n moves the input pointer to a specific column number
- the+n moves the input pointer forward to a column number relative to the current position
- **Format:**
  INPUT @n *variable informat.;*
- **Format:**
  INPUT +n *variable informat.;*

@n is an absolute pointer control.
@ moves to column n, the first column to be read.
When reading a record, the @n pointer can be moved forward and backward.

+n moves the input pointer forward to a column number relative to the current position.
The notation +(-n) will move the pointer control backwards.

8.1.2 Informats

Informats are used to explain how the raw data is read. The $w. informat allows character data to be read. The dollar sign indicates character only data. The w represents the field width, or number of columns, of the data. The period ends the informat.

The w.d format allows standard numeric data to be read. The w represents the field width, or number of columns, of the data.
If a decimal point exists with the raw data, that acts as one decimal. The period acts as a delimiter. The optional d specifies the number of implied decimal places (not necessary if the value already has decimal places).

The COMMAw.d will read nonstandard numeric data, removing any embedded:

- blanks
- commas
- dashes
- dollar signs
- percent signs
- right parentheses
- left parentheses.

8.1.3 Record Formats

How data is read by column input and formatted input processes can be affected by the record format of the external file. Two common record formats are fixed-length records and variable-length formats.

A fixed-length record format has a predetermined number of columns. The end-of-record marker exists after the last column. The typical fixed-length record will have a length of 80 columns.

A variable-length record format has an end-of-record market after the last field of each record. Variable-length records containing fixed-field data might have values which are shorter than others or values that are missing.

With varied data, the end of column can conflict with the end-of-record marker causing errors. A PAD option can be used to bypass common problems with reading fixed-field data in variable-length records. The PAD option will add blanks to each record to give all data lines the same length.

The default value of the maximum record length is determined by the operating environment. The maximum record length can be changed using the LRECL=option in the INFILE statement.

8.2 Reading Free Format Data

In free format records, the fields are often separated by a delimiter, such as blanks or the pound sign.

A List input can read standard and nonstandard data in a free-format record:

- Use an INPUT statement
- Format:
  INPUT *variable <$>;*
    - *variable* is the name of the variable to be read
    - $ identifies the variable as a character variable.

By default, List input does not have specified column locations, so:

- all fields must be separated by at least one delimiter
- fields cannot be skipped or re-read
- the order for reading fields is from left to right.

8.2.1 Processing List Input

List input processing will scan for values instead of reading from specific columns.

- SAS reads the first field until it encounters a delimiter
- The delimiter indicates the end of the field, so the data value is assigned to the first variable in the program data vector
- Then the scan continues until the next delimiter is encountered
- The data value for the next variable is assigned in the program data vector
- This continues until all the fields have been read and values are assigned to variables.

The data set can be displayed using the PRINT procedure.

Blanks as delimiters are easy to read. Other delimiters need to be explicitly identified.

To specify a delimiter:

- Use an INFILE statement and DLM-option
- Format:
  INFILE *file-specification* DLM=*delimiter(s);*
  - *file-specification* takes the form of fileref from a previously defined file reference or 'filename' to point to the actual name and location of the file
  - *delimiter(s)* specifies a delimiter for list input.

Delimiters can be specified in two ways:

- A 'list of delimiting characters' will specify one or more characters (must be enclosed in quotation marks)
- character-variable specifies a character variable to be a delimiter.

A delimiter should be the same as a character within the field values.

When the variable values are sequential and separated by a delimiter, variables can be expressed and specified in a range.

By default several limitations exist on the type of data that can be read using list input:

- Character values that are longer than eight characters will be truncated
- Data must be in standard numeric or character format
- Character values cannot contain embedded delimiters
- Missing numeric and character values must be represented by a period or some other character.

8.2.2 Reading Missing Values

Missing values can exist anywhere in the data set. If missing values occur at the end of the record, the MISSOVER option in the INFILE statement can be used to read them. The MISSOVER option will prevent the SAS from going to another record if values cannot be found for every specified variable in the current line.

MISSOVER only works with missing values at the end of a record. To begin to read missing values in the beginning or middle of the record, the DSD option in the INFILE statement can be used.

DSD changes how delimiters are treated when using a list input:

- by setting the default delimiter to a comma
- treating two consecutive delimiters as a missing value
- removing quotation marks from values.

If multiple delimiters are used in the original file, they can be identified by using the DLM=option.

Modifying List inputs allows it to be more versatile.
Two modifiers can be used:

- the ampersand is used to read character values that contain embedded blanks
- the colon is used to read nonstandard data values and character values longer than eight characters.

8.2.3 Managing Length

The default length of character values is 8. Variables that are longer than 8 are truncated when written to the program data vector.

Using a LENGTH statement before the INPUT statement will define the length and type of the variable.

8.2.4 Creating Free-Format Data

To create free-format data:

- Use a PUT statement
- Format:
  PUT *variable <: format>;*
    - *variable* is the name of the variable to be written
    - a colon precedes a format
    - *format* specifies a format to use when writing variables.

Using a DLM=option in the FILE statement can be used to be create a character-delimited raw data file. If special characters are required in the data format, use the DSD=option in the FILE statement, including quoted strings.

To create a simple raw data file:

- Use a PROC EXPORT statement
- Format:
  PROC EXPORT DATA=*SAS-data-set;*
     OUTFILE=*filename* <DELIMITER='*delimiter*'>;
  RUN*;*
    - *SAS-data-set* names the input SAS data set
    - *filename* identifies the complete path and file name of the output
    - *delimiter* identifies the delimiter used to separate columns in the output file.

8.3 Reading Data and Time Values

SAS stores a date value as the number of days from January 1, 1960 to a given date. SAS stores a time value as the number of seconds since midnight.

The SAS datetime value is the number of seconds between midnight on January 1, 1960 and a given data and time.

Data and Time informats are used to read data and time expressions and convert them to SAS data and time values. Like other informats, they consist of:

- a name
- a field width
- a period delimiter.

When specifying an informat:

- Use an INPUT statement
- Format:
  INPUT *<pointer-control> variable informat.;*
  - *pointer-control* places the input pointer in the appropriate column
  - *variable* is the name of the variable to be created
  - *informat* is the special instruction specifying how SAS reads raw data.

Some common data informats:

MMDDYYw.	-	11/15/09
Dates.	-	15Nov09
MMDDYYw.	-	11-15-09
YYMMDDw.	-	09/11/15

MM will always represent the month and be an integer between 01 and 12. DD will always represent the day and be an integer between 01 and 31. YY is the year, which can be represented YYYY. w is the field width.

Date fields can be separated by blanks, hyphens, and slashes.
The field width should be large enough to include all delimiters.

- Time informat is expressed as hh:mm:ss:ss.
- The hh represents hours and be an integer between 00 and 23.
- The mm represents minutes and be an integer between 00 and 59.
- The ss:ss represents seconds and hundredths of seconds and is optional.

The WEEKDATEw. format will display the day of the week, month-name, day and year.

The WORDDATEw. format will write the data in the form month-name, day, and year.

9 Working with Observations

In the simplest form, the information for a single observation will be found in a single record.
In many cases, the information for a single observation can be found in multiple records.
To access this information:

- write multiple INPUT statement to read each record
- write one INPUT statement contain a line pointer control to specify the record(s) being read.

9.1 Creating One Observation From Multiple Records

9.1.1 Line Pointer Controls

When SAS reads raw data, its position is tracked using an input pointer. Column pointer controls are used to determine the column position of the input pointer. The input pointer can also be positioned on a specific record using a line pointer control.

Two types of line pointer controls exist:

- the forward slash specifies the line location relative to the current one
- the #n specifies an absolute number of the line for moving the pointer.

9.1.2 Reading Multiple Records

To read multiple records sequentially:

- use the forward slash line pointer
- the / line pointer control will only move the input pointer forward
- the / line pointer control is specified after instructions for reading values in the current record.

When a DATA step executes, the values of the first record are read and the input pointer is moved by the line pointer control to the second record. The second record is read and the second line pointer control moves the input pointer to the third record. This continues until the INPUT statement is complete.

The values read are written to the data set as an observation. Control returns to the top of the DATA steps and the variable values are reinitialized to missing. The process continues starting where the last iteration left off and continues until all records have been read. The number of records for each observation must be the same.

To read records non-sequentially:

- The #n line pointer control specifies the absolute number of the line where the input pointer will be moved
- This absolute condition allows records to be read in any order
- The #n line pointer control must be specified before any instructions for reading values in a record.

A SAS program can use the forward slash line pointer control and #n line pointer control together to read records sequentially and non-sequentially.

9.2 Creating Multiple Observations From Single Records

In some cases, the information for multiple observations can be found in a single record.
To manage the control for reading a record, a line-hold specifier is required.
SAS provides two line-hold specifiers:

- a trailing at sign (@) will hold the input record for the execution of the next INPUT statement
- a double trailing at sign (@@) holds the input record for the execution of the next INPUT statement, even across iterations of the DATA step.

Normally, each time a DATA step executes, the INPUT statement reads a new record. When @@ is used:

- to read multiple SAS observations from single data line
- the data line is held in the input buffer across multiple executions of the DATA step
- the @ pointer control, column input, nor the MISSOVER option should be used.

A record being held by @@ will not be released until:

- the input pointer moves past the end of the record
- an INPUT statement that has no line-hold specifier executes.

With a single @, reading records:

- will allow the next INPUT statement to read from the same record
- release the current record when a subsequent INPUT statement executes without a line-hold specifier.

10 Creating Reports

10.1 Basic Reports

The PROC PRINT statement can be used to create a printout, or report, of the information in a data set. The report can be enhanced with additional statements and options.

To print the contents of a SAS data set:

- Use the PROC PRINT statement
- Format:
  LIBNAME *libref 'SAS-data-library'*
     PROC PRINT data=*SAS-data-set*;
  RUN;
    - *libref* is 1 to 8 characters long beginning with a letter or underscore and containing only letters, numbers, or underscores
    - *SAS-data-library* is the name of the library where files are stored
    - *SAS-data-set* is the name of the SAS data set to be printed.

Default layout for a report includes:

- all observations and variables are printed
- observations numbers are listed in the far left column
- variables appear in the order they occur in the data set.

Statements and options for changing report characteristics

SUM *variable(s)*	Produce column totals for numeric values
VAR *variable(s);*	Select the variables to include in the report and the order they appear
PROC PRINT data=*libref* NOOBS	Observation numbers are not displayed
ID *variable(s);*	Identifies one or more variables that will replace the

	OBS column. If a variable is listed in an ID statement and a VAR statement, the output will have two columns for the same variable
WHERE *where-expression*	Identifies the conditions for selecting observations in the report. The where-expression can be any SAS expression
LABEL *variable=new-variable-name*	Change the label name of a variable

10.1.1 WHERE Expressions

Any variable found in the SAS data set can be specified in the WHERE statement, not just variables specified in the VAR statement. Either character or numeric variables can be included in an WHERE statement.

Conditions based on the value of a character variable must:

- be enclosed in quotation marks
- must be written exactly as it appears in the data set.

The operators, CONTAINS and ?, can be used with the WHERE statement to select observations with a specific substring.

Multiple conditions can be used to select observations using logical operators. When testing for multiple values of the same variable:

- The variable name is specified in each expression
  ```
  WHERE variable=value1 OR variable=value2;
  ```
- The use of the IN operator
  ```
  WHERE variable IN (value1, value2);
  ```
- Control the evaluation of compound expression, use parentheses.

10.1.2 Sorting Data

Default reports list observation in the order they appear in the data set. To sort the report based on the values of a variable, use the PROC SORT statement.

The SORT procedure will:

- rearrange observations in the SAS data set
- create a new SAS data set containing the rearranged observations
- replace the original SAS data set
- sort on multiple variables
- sort in ascending or descending order
- treats missing values as the smallest value
- not generate printed output.

To perform a basic sort of variables:

- Use the PROC SORT statement
- Format:

  PROC SORT DATA=*SAS-data-set* <OUT=*SAS-data-set*>*;*
  BY <DESCENDING> *BY-variable(s);*
  RUN;

  - DATA= specifies the data set to be read
  - OUT= specifies the output data set that contains the data in sorted order
  - *BY-variable(s)* specifies one or more variables used to sort the data
  - DESCENDING identifies if the input data set should be sorted from largest to smallest. The absence of DESCENDING has the input data set sorted in ascending order.

Without the OUT= option, the data set specified is permanently sorted.

10.1.3 Creating Subtotals

Creating column totals for numeric variables is done using SUM statement. To subtotal numeric variables, use a BY statement in conjunction with the SUM statement.

To create a total and subtotals for numeric variables:

- Use the SUM statement and the BY statement
- Format:
  SUM *variable(s);*
  BY <<DESCENDING> <*BY-variable(s)*>>
  <NOTSORTED>;
  - *variable(s)* specifies one or more variables, separated by blanks.
  - *BY-variable(s)* specifies variables to be used to form BY groups. More than one variable should be separated by blanks
  - DESCENDING identifies if the input data set should be sorted from largest to smallest. The absence of DESCENDING has the input data set sorted in ascending order
  - NOTSORTED specifies that the observations are not necessarily sorted in any order.

Variables listed in the SUM statement are not required to be specified in the VAR statement.

10.1.4 Creating a Customized Layout

Using the ID statement in conjunction with the ID and SUM statements will show the BY variable heading only once:
If the variable specified by the IN statement is the same as the BY statement, then:

- The OBS column is suppressed
- The ID/BY variable is printed in the far left column
- Each ID/BY value is printed at the start of each BY group and on the same line as the group's subtotal.

Each BY group can be printed on a separate page using the PAGEBY statement:

- Format:
  PAGEBY *BY-variable;*

To double space the report, use the DOUBLE option in the PROC PRINT statement.

10.1.5 Titles and Footnotes

Reports can be created so they are easy to interpret, including:

- adding titles and footnotes
- replacing variable names with descriptive labels
- formatting variable values.

Up to 10 titles with procedure output can be specified using the TITLE statement before the PROC step.
Up to 10 footnotes can be specified using the FOOTNOTE statement before the PROC step.

- Format:
  TITLE <*n*> *'text';*
  FOOTNOTE <*n*> *'text';*
  PROC PRINT *SAS-data-set;*
  - *n* is a number from 1 to 10 which specifies the title or footnote
  - *'text'* is the title or footnote to be displayed.

- The maximum length is dependent on the operating environment and the value of the LINESIZE= option
- TITLE is like TITLE1; FOOTNOTE is like FOONOTE1
- The default tile is The SAS System
- There is no default footnote.

Blank lines can be created in the title or footnote by skipping title expressions; such as title**1**, title**3**.

When multiple titles occur:

- In HTML output, the lines appear consecutively without extra spacing
- In SAS listing output, lines are centered by default.

When multiple footnotes occur:

- The FOOTNOTE statement with the largest number appear on the bottom line
- In HTML output, the lines appear consecutively without extra spacing
- In SAS listing output, lines are centered by default.

TITLE and FOOTNOTE statements are global statements and are in place until they are modified, canceled, or the SAS session ends.

When redefining a title or footnote, all higher-numbered titles or footnotes are canceled. A null TITLE or FOOTNOTE statement has no number or text.

10.1.6 Assigning Descriptive Labels

Columns can be labeled with descriptive text.
To label columns, use:

- the LABEL statement to assign a descriptive label to a variable
- the LABEL option in the PROC PRINT statement to specify the labels to be displayed

- Format:
  ```
  LABEL variable1='label1';
        variable2='label2';
         ... ;
  ```

Variables can be assigned separate LABEL statements or be grouped with several variables assigned a single LABEL statement.

10.1.7 Assigning Formats

To associate formats with variables, use the FORMAT statement. Formats only affect the output of the data values, not the actual data values stored.

To format a variable:

- **Format:**
  FORMAT *variable(s) format-name;*
    - *variable(s)* is a name of one or more variables whose values are to be written according to a particular pattern
    - *format-name* specifies a SAS or user-defined format for the values.

Variables can be assigned separate FORMAT statements or be grouped with several variables assigned a single FORMAT statement.

Common SAS formats:

w.	Rounded to the nearest integer in w spaces.
w.d	Rounded to d decimal places in w spaces.
$w.	Character values in w spaces.
COMMAw.d	Contains commas and decimal places.
DOLLARw.d	Contains dollar signs, commas and decimal spaces.
DATEw.	Date values in the form 12/12/10 (DATE7.) or 12DEC2010 (DATE9.)
MMDDYYw.	Date values in the form 12DEC10 (DATE8.) or 12/12/2010 (DATE10.)

All SAS formats specify a total field width (w).
The field width includes the numeric digits and commas.

Stored Value	4060
Desired Format	COMMAw.d

Displayed Value	4,060
Positions Displayed	5
FORMAT Statement	format variable comma5.0;

The number of decimal places used for a numeric decimal can be specified (D).

Stored Value	4060
Desired Format	COMMAw.d
Displayed Value	4,060.00
Positions Displayed	8
FORMAT Statement	format variable comma8.2;

Currency is displayed with appropriate dollar signs, commas, and decimal spaces.

Stored Value	4060
Desired Format	DOLLARw.d
Displayed Value	$4,060.00
Positions Displayed	9
FORMAT Statement	format variable dollar9.2;

To temporarily assign a label or format to the data output, use the LABEL or FORMAT statements within the PROC PRINT step. To permanently assign a label or format to the data , use the LABEL or FORMAT statements within the DATA step.

10.2 Advanced Reporting

To create advanced forms of reports, use the PROC REPORT statement.
Types of customization allow:

- Request separate subtotals and grand totals
- Calculate columns
- Create and store report definitions.

The PROC REPORT statement can be used:

- In a windowing mode with prompts to guide reporting building
- In a windowing mode without the prompts
- In a nonwindowing mode.

10.2.1 Default List Reports

To create a basic list report:

- Use the PROC REPORT statement
- Format:
  PROC PRINT <data=*SAS-data-set*> <*options*>;
  RUN;
    - *SAS-data-set* is the name of the SAS data set to be used for the report
    - *options* provides the report in a window or non-windowing environment.

The COLUMN statement identifies the variables displayed in the report and their order of appearance.

- Format:
  COLUMN *variable(s);*

The WHERE statement identifies the observations, or rows, to be selected based on a condition.

10.2.2 Defining Variables

Reports can be enhanced by:

- defining how variables can be used in the report
- assigning formats to variables
- specifying column headings and widths
- justifying the variable values
- justifying column headings
- changing the order of rows.

The DEFINE statement is used to describe the use and display of variables.
To define variables:

- Use the **DEFINE** statement
- **Format:**
  DEFINE variable/ *<usage> <attribute(s)> <option(s)> <justification> <"column heading">;*
  - *variable* is the name of the variable to be defined
  - *usage* specifies how a variable is to be used. Valid options are ACROSS, ANALYSIS, COMPUTED, DISPLAY, GROUP, and ORDER
  - *attribute(s)* specifies the attributes of the variables, including FORMAT=, WIDTH=, and SPACING=
  - *option(s)* specifies formatting options, including DESCENDING, NOPRINT, NOZERO, and PAGE
  - *justification* specifies the justification of the column (CENTER, LEFT, or RIGHT)
  - *"column heading"* specifies the label for the column.

Format controls the appearance of data in the report. Any SAS or user-defined format can be used with FORMAT=.

By default, the column width is the length of the variable for character variables or 9 for numeric variables.

- The WIDTH=attribute can be used to specify the width for the columns in the report
- The value of WIDTH can be 1 to the value of the LINESIZE= system option
- WIDTH= attribute has no effect on HTML output.

Spacing of columns can allow reports to be easier to read. Defining spacing allows the number of blank characters between the selected column and the column immediately to the left to be specified.

- The default spacing is 2
- The spacing can be redefined using SPACING= attribute
- The SPACING= attribute has no effect on HTML output.

Column headings can be changed by enclosing the heading text in quotation marks. How words break in column headings can be controlled using a split character. To use a split character:

- use the default slash (/)
- use the SPLIT= option and define a split character.

By default, PROC REPORT will left-justify character variables and right-justify numeric variables. To define a different justification, specify the desired option (right, left, center) in the DEFINE statement. The option will justify the column headings and the values within the column width.

The appearance of the headings found in the list report can be changed using two options:

- HEADLINE underlines all column headings and the spaces between them
- HEADSKIP creates a blank line beneath all column headings or after the underline if the HEADLINE option is used.

These options have no effect on HTML output.

10.2.3 Variable Usage

How variables are used will change the layout of the report.
Each variable can be used:

- DISPLAY – default for character variables
- ORDER
- GROUP
- ACROSS
- ANALYSIS – default for numeric variables
- COMPUTED.

Display variables don't affect the order of rows in the report. If one or more display variables are in the report, a detail row for each observation is read from the data set. A detail contains a value for each display variable.

Order variables determine the order of rows in the report. The detail rows are ordered according to their formatted values. By default, the order is ascending: to change this, use the DESCENDING option.

Summary reports are possible using group variables. A group variable will group the detail rows in a report according to their formatting value. The PROC REPORT will consolidate the relevant observations into one row.

To create a summary report, all variables in the report have to be defined as group, analysis, across, or computed. If a variable in the report is not defined or defined as display or group, a group variable is displayed as an order variable.

The default statistic for analysis variables is SUM. To associate a statistic with an analysis variable, the statistic should be specified as an attribute in the DEFINE statement.

Possible statistical attributes:

CSS	Corrected sum of squares
CV	Coefficient of variation
MAX	Maximum value
MEAN	Average
MIN	Minimum value
N	Number of observations with nonmissing values
NMISS	Number of observations with missing values
PCTN	Percentage of a cell or row frequency to a total frequency
PCTSUM	Percentage of a cell or row sum to a total sum
PRT	Probability of a greater absolution value of student's *t*
RANGE	Range
STD	Standard deviation
STDERR	Standard error of the mean
SUM	Sum
SUMWGT	Sum of the Weight variable values
T	Student's *t* for testing the hypothesis that the population mean is 0
USS	Uncorrected sum of squares
VAR	Variance

Across variables are functionally similar to group variables. Groups are created horizontally, not vertically, with across variables. Analysis variables will have all values for a variable summed when using across variables.

Computed variables can be associated to character or numeric variables. Computed variables are not in the input data set, and they are not added to the input data set by the PROC REPORT. The use of a computed variable cannot be changed.

Computed variables can be added in a nonwindowing environment:

- Include the computed variable in the COLUMN statement
- Define the usage as COMPUTED
- Compute the value of the variable in a compute block associated with the variable.

10.3 Descriptive Statistics

Descriptive statistics are used to summarize large amounts of data. The type of descriptive statistics required and the SAS procedure used is dependent on what needs to be summarized.
Continuous data values or discrete data values can be summarized.

10.3.1 Computing Numeric Variables

The MEANS procedure is used to provide mean, minimum, and maximum and other data summarization to control the output of the report.

To create a MEANS procedure:

- Use **PROC MEANS** statement
- **Format:**
  PROC MEANS <DATA=*SAS-data-set*>
  <statistic-keyword(s)> <option(s)>;
  RUN;
    - *SAS-data-set* is the name of the SAS data set used
    - *statistic-keyword(s)* identifies the statistics to compute
    - *options* will control the content, analysis, and appearance of the output.

In its simplest form, PROC MEANS will print for every numeric variable in a data set:

- the number of nonmissing values (n-count)
- the mean
- the standard deviation
- the minimum value
- the maximum value.

To limit the output, the use of statistic keywords will focus the output.

DESCRIPTIVE STATISTICS	
CLM	Two-sided confidence limit for the mean
CSS	Corrected sum of squares
CV	Coefficient of variation
KURTOSIS / KURT	Kurtosis
LCLM	One-sided confidence limit below the mean
MAX	Maximum value
MEAN	Average
MIN	Minimum value
N	Number of observations with nonmissing values
NMISS	Number of observations with missing values
RANGE	Range
SKEWNESS /SKEW	Skewness
STDDEV /STD	Standard deviation
STDERR / STDMEAN	Standard error of the mean
SUM	Sum
SUMWGT	Sum of the Weight variable values
UCLM	One-sided confidence limit above the mean
USS	Uncorrected sum of squares
VAR	Variance

QUANTILE STATISTICS	
MEDIAM / P50	Median or 50th percentile
P1	1st percentile
P5	5th percentile
P10	10st percentile
Q1 / P25	Lower quartile or 25th percentile
Q3 / P75	Upper quartile or 75th percentile
P90	90st percentile
P95	95st percentile
P99	99st percentile
QRANGE	Difference between upper and lower quartiles: Q3-Q1
HYPOTHESIS TESTING	
PROBT	Probability of a greater absolute value for the *t* value
T	Student's *t* for testing the hypothesis that the population mean is 0

10.3.2 Formatting Statistics

By default, PROC MEANS uses the best format to display values which can lead to unnecessary decimal places. MAXDEC= can limit the number of decimal places displayed.

By default, statistics are generated for every numeric variable in a data set. To focus on specific variables, a VAR statement can be used.

By default, the statistics will be applied to all observations in a data set. To apply statistics to groups of observations, add a CLASS statement.

The values of CLASS variables are used only to categorize data, therefore, statistics are not generated for CLASS variables. They can be either character or numeric, but must have a limited number of discrete values to be grouped meaningfully. The order of the variables in the CLASS statement determines their order in the output.

BY statements can specify variables for categorizing like CLASS statements. Differences include:

- BY processing requires that data is already sorted or indexed. The SORT procedure must be used before using PROC MEANS
- The layout of group results are different: A CLASS statement will create a single table, while a BY statement will create multiple tables.

10.3.3 Creating Summarized Data Sets

A SAS data set can be created that contains only a summarized variable.
To create summarized DATA set using PROC MEANS:

- Use PROC MEANS statement and OUTPUT statement
- Format:
  PROC MEANS <DATA=*SAS-data-set*> *<statistic-keyword(s)> <option(s)>*;
  OUTPUT OUT=*SAS-data-set-out <statistic-keyword=variable-name(s)>;*
  RUN;
    - *SAS-data-set* is the name of the SAS data set to be used
    - *statistic-keyword(s)* identifies the statistics to compute
    - *options* will control the content, analysis, and appearance of the output
    - *SAS-data-set-out* is the name of the output SAS data set
    - *statistic-keyword* identifies the summary statistic to be written out
    - *variable-name(s)* identifies the names of the variables created to contain the values of the summary variables.

When the statistic-keyword= option is not used, the summary statistics produced will be N, MEAN, STD, MIN, and MAX for all numeric variables listed in the VAR statement or all numeric variables is no VAR statement exists.

The keyword for the statistics must be specified and then all the variables to be computed. If a VAR statement exists, the variables must be listed in the same order as the variables in the VAR statement.

Summarized output data set can be created using the SUMMARY procedure. The PROC SUMMARY uses the same code to produce output data as PROC MEANS. The difference between PROC MEANS and PROC SUMMARY is the use of defaults: PROC MEANS produces a report by default, PROC SUMMARY must include a PRINT option.

10.4 Creating Frequency Tables

To describe the data by reporting the distribution of variable values, the FREQ procedure should be used. The FREQ procedure is a descriptive procedure and a statistical procedures. One-way and n-way, or crosstabulation, frequency tables can be created.

The FREQ procedures can include several statements and options for controlling output.
To create a basic frequency report:

- Use **PROC FREQ** statement
- **Format:**
  PROC FREQ <DATA=*SAS-data-set>;*
  RUN;
  - *SAS-data-set* is the name of the SAS data set to be used.

By default, a one-way table is created with the frequency, percent, cumulative frequency, and cumulative percent of every value of all variables in a data set.

- Frequency is the number of observations with the desired value
- Percent is the frequency of the value divided by the total number of observations
- Cumulative frequency is the sum of the frequency counts of the value and all other values above it in the table
- Cumulative percent is the cumulative frequency of the value divided by the total number of observations.

By default, the FREQ procedure will create frequency tables for every variable in the data set. Frequency distributions work best with variables that can be categorized and whose values are best summarized by counts.

To identify the variables to be processed by the FREQ procedure:

- Use TABLES statement
- Format:
  TABLES *variable(s);*

10.4.1 Creating N-Way Tables

Crosstabulation is used when two distinct, but related, variables are compared. The most basic form of crosstabulation is a two-way table.

To create an n-way table:

- Use PROC FREQ statement and TABLES statement
- Format:
  PROC FREQ <DATA=*SAS-data-set>;*
  TABLES *variable-1* variable-2 <*.. variable-n>;*
  RUN;
    - *SAS-data-set* is the name of the SAS data set to be used
    - *variable-1* identifies table rows
    - *variable-2* identifies table columns
    - *variable-n* identifies a multi-way table.

When specifying crosstabulation, PROC FREQ produces tables with cells that contain:

- cell frequency
- cell percentage of total frequency
- cell percentage of row frequency
- cell percentage of column frequency.

When creating n-way tables, a series of two-way tables are produced with a table for each level of the variables.

10.4.2 Table Layout

To read n-way tables, the order of variables is an important consideration. The last two variables of the TABLES statement will become the two-way rows and columns. A table will be created for each value of the of the variables preceding the last two variables.

A CROSSLIST option will display crosstabulation tables in ODS column format. CROSSLIST creates a table with a customizable table definition. A table definition is customized using the TEMPLATE procedure.

- Format:
  TABLES *variable-1* variable-2* <*.. *variable-n*> /CROSSLIST*;*

A LIST option will allow bulky, complex crosstabulations to be read as a continuous list - great for n-way tables that have three or more variables specified. It eliminates row and column frequencies and percents

- Format:
  TABLES *variable-1* variable-2* <*.. *variable-n*> /LIST*;*

The output of the FREQ procedures can be limited to a few specific statistics:

- NOFREQ suppresses cell frequencies
- NOPERCENT suppresses cell percentages
- NOROW suppresses row percentages
- NOCOL suppresses column percentages.

10.5 Producing HTML Output

SAS uses Output Delivery System (ODS) statements to generate HTML output.
With ODS, HTML output in any operating environment can be created, customized, and managed through programming statements.

10.5.1 Output Delivery System

Output can be created in a variety of formats with ODS:

- HTML output
- Output data set of procedures results
- Traditional SAS listing output.

When ODS statements are submitted and the output is created by the SAS program:

- ODS creates the output in the form of output objects; each output object containing:
  - Data component - the results of a procedure or DATA step
  - Table definition – information about how the results are rendered.
- The output object is sent to specified ODS destination(s) and creates the formatted output controlled by the destination.

ODS destinations supported include:

- HTML – output formatted in HTML
- Listing – output formatted like traditional SAS procedure output
- Markup Language Family – output formatted in markup languages, such as XML
- ODS Document – hierarchy of output objects
- Output – SAS data sets
- Printer Family – output formatted in PS, PDF, or PCL files
- RTF – output formatted in RTF format using Microsoft Word.

For each type of formatted output created, an ODS statement is required to open the destination and another ODS statement to close the destination. The default destination is Listing.

To open and close ODS destinations:

- Use **ODS** statement
- **Format:**
  ODS *open-destination;*
  ODS *close-destination* CLOSE *;*
    - *open-destination* is the keyword and any required options for the output type to be created
    - *close-destination* is the keyword for the type of output.

Since the default destination is Listing, it is considered always open. Best practices would close the listing destination at the beginning of the program.

If multiple ODS destinations are open concurrently, they can be closed at the same time using the keyword _ALL_.

10.5.2 Creating Simple HTML

To create HTML output:

- Use ODS HTML statement
- Format:
  ODS HTML BODY=*file-specification;*
  ODS HTML CLOSE *;*
    - *file-specification* identifies the file containing the HTML output. The specification can be:
        - an HTML filename
        - a fileref
        - a SAS catalog entry.

ODS HTML statements can be used to direct the output from multiple procedures to the same HTML file.

To create HTML output with a table of contents:

- Use ODS HTML statement
- Format:
  ODS HTML
  - BODY=*body-file-specification;*
  - CONTENTS=*contents-file-specification;*
  - FRAME=*frame-file-specification;*

  ODS HTML CLOSE *;*
    - *body-file-specification* is the name of the HTML file containing the procedure output
    - *contents-file-specification* is the name of the HTML file containing a table of contents with links to the procedure output
    - *frame-file-specification* is the name of the HTML file connecting the table of contents with the body file

- A CONTENTS= option is required if FRAME= is used
- A table of contents will contain a numbered heading for each procedure that creates output
- Links are generated between files by using HTML filenames specified in the ODS HTML statement.

The URL=suboption in the BODY or CONTENTS file specification will allow an URL to be used for all links that is created to the file. Either relative URLs or absolute URLs can be specified.

PATH=option will allow the specification of a location where HTML output can be store and assists in streamlining the ODS HTML statement.

STYLE=option can be used to change the appearance of the HTML output, and valid SAS or user-defined style definition can be used.

11 Advanced Work with Variables

11.1 User-Defined Formatting

Raw data or data found in a basic SAS data set may not be easily read in its basic form requiring additional formatting to clarify the information.

A common occurrence is the existence of coded information: a set of numerical expressions used to identify meaningful information. PROC FORMAT can be used to replace the code with the information it represents.

To define a specific format:

- Use PROC FORMAT statement
- Format:
  PROC FORMAT *<options>;*
  options include:
  - LIBRARY=libref identifies the SAS data library which contains the catalog storing user-defined formats
  - FMTLIB will print the contents of a format catalog

All user-defined formats are stored in a format catalog. If one doesn't exist, SAS will automatically create one named Work.Formats. Work.Formats is a temporary library that exists until the end of the SAS session.

A permanent format catalog named Formats can be used when specified in the LIBRARY=option of the PROC FORMAT statement. To use the Formats library, a LIBNAME statement is required to associate the libref to the permanent library. When a permanent format is associated with a variable, the libref is used to reference the location of the format catalog.

11.1.1 Creating User-Defined Formats

To define format for displaying one or more values:

- Use VALUE statement
- Format:
  VALUE *format-name*
  range1='label1'
  range2=label2'
  ...;
    - *format-name* is the name of the format that is being created
    - *range* specifies one or more variable values and a character string or existing format
    - *label* is a text string enclosed in quotation marks.

A VALUE format name must:

- begin with a dollar sign if the format is applied to character data
- not be longer than eight characters
- not be the name of an existing SAS formatting
- not end in a number
- not end in a period when in the VALUE statement.

A VALUE range will identify:

- a single value
- a range of numeric values
- a range of character values enclosed in quotation marks
- a list of unique values separated by commas.

Character values must be enclosed in quotation marks and match the variable's values. The format's name must start with a dollar sign.

Numeric values do not need to be enclosed in quotation marks. The format's name does not require a dollar sign.

Non-inclusive ranges can be specified:

- using the "less than" symbol to avoid overlapping, as in a-<b
- using the keywords LOW and HIGH to identify lower and upper limits, as in low-<b and a-<high
- using the keywords OTHER to label missing values.

Labels used to display ranges must be:

- enclosed in quotation marks (double quotations if an apostrophe is part of the label)
- limited to 256 characters.

Multiple formats can be defined by using multiple VALUE statements in a single PROC FORMAT statement.

Each VALUE statement will define a different format.

11.1.2 Using User-Defined Formats

Permanent, user-defined formats are stored in a format catalog.

When using a user-defined format, SAS will search in two libraries:

- the temporary library, Work
- a permanent library, Library

SAS uses the first instance of the format found.

Association of formats with variables is done using a FORMAT statement:

- use the same format name in the FORMAT statement as in the VALUE statement
- end the format name in the FORMAT statement with a period.

11.1.3 Displaying User-Defined Formats

Top display a list of all the formats in the format catalog with the descriptions of their values, the keyword FNTLIB can be added to the PROC FORMAT statement.

The name, range, and label for each format is provided, as well as:

- the length of the longest value
- number of values defined by this format
- version of SAS the format is compatible with
- the data and time the format was created.

11.1.4 Creating Variables

Anew variable can be created to accumulate the values of multiple variables.
To create an accumulator variable:

- Use **SUM** statement
- **Format:**
  variable+expression;
    - *variable* is the name of the accumulator variable
    - *expression* is any valid SAS expression.

An accumulator variable must be numeric. The variable is automatically set to 0 before reading the first observation.

The sum statement will add the results of the expression on the right side of the plus sign to the numeric variable on the left. The numeric variable will retain the new value in the program data vector until the next observation is read.

By default, an accumulator variable is initialized to 0. To initialize to a different number, use the RETAIN statement.

The RETAIN statement will:

- assign an initial value to a retained variable
- prevent variables from being initialized with each execution of the DATA step

To initialize a variable to a value other than 0:

- Use RETAIN statement
- Format:
  RETAIN *variable initial-value;*
  - *variable* is the name of the variable whose value will be retained
  - *initial-value* is the initial numeric or character value for the variable.

Restrictions to the RETAIN statement include:

- no effect on variables which are read with SET, MERGE, and UPDATE statements
- the retained variable will be initialized to missing of no initial value is specified
- the RETAIN statement is a compile-time only statement to create variables that do not already exist.

11.1.5 Grouping Variables

Variables can be assigned values conditionally.
To conditionally assign a value:

- Use IF-THEN statement
- Format:
  IF *expression* THEN *statement;*
  - *expression* is any valid SAS expression
  - *statement* is any executable SAS statement.

The assignment statement is executed when the condition is true; if false the value of the variable is missing. Any of the comparison of logical operators can be used in an IF-THEN statement. Logical comparisons are enclosed in parentheses to be evaluated.

IF-THEN ELSE statements allow alternative actions to be applied to the condition. Multiple IF-THEN ELSE can be used to provide numerous alternatives. SAS will execute the IF-THEN statements until a true statement is found.

The length of the variable can vary based on the expressions used unless it is explicit specified using the LENGTH statement.

- Format:
  LENGTH *variable(s) <$> length;*

The dollar sign is required if the variable is a character variable.

Some observations may be unwanted; to remove them use the DELETE statement.

- Format:
  DELETE ;

To conditionally delete an observation:

- Use IF-THEN statement and DELETE statement
- Format:
  IF *expression* THEN *DELETE*
  If the *expression* is:
    - TRUE, the DELETE statement executes
    - FALSE, the DELETE statement does not execute and the processing continues.

To read and process fields that are not kept in the data set, use the DROP= and KEEP= data set options:

- Format:
  (DROP *variable(s))*
  (KEEP *variable(s))*

The KEEP option is used when there are more variables dropped than kept.

Exclusion of variables can be done using statements instead of data set options; the differences are:

- DROP statements cannot be used in the SAS procedure steps
- The DROP statement applies to all output data sets named in the DATA statement
- To exclude variables in some data sets but not all, use the DROP data set options and associate it to the individual data sets to impact.

Another method for conditionally assigning values is to sue SELECT groups.

To conditionally assign values using SELECT:

- Use SELECT statement
- Format:
  SELECT <*select-expression)>;*
      WHEN-1 *(when-expression-1 <..., when-expression-n>) statement;*
      WHEN-n *(when-expression-1 <..., when-expression-n>) statement;*
      <OTHERWISE *statement;*>
  END;
    - SELECT begins the SELECT group
    - *select-expression* is the SAS expression that is evaluated
    - WHEN identifies the SAS statements to be executed when a condition is true
    - *when-expression* identifies any SAS expression, value. There must be at least one expression
    - statement is any executable SAS statement
    - OTHERWISE specifies a statement to be executed when all WHEN statement prove false (optional)
    - END terminates the SELECT group processing.

Conditional processing can be done on groups of statements rather than single SAS statements by creating DO groups.
To create a DO group:

- Use DO statement
- Format:
  DO;
      SAS statements
  END;
    - DO statement begins the processing
    - SAS statement are called a DO group and executed as a unit
    - END terminates the DO group processing.

DO groups can be used within IF/THEN.ELSE statements and SELECT groups.

Three forms of DO statements:

- The iterative DO statement will execute any statement between DO and END repetitively
- The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true
- The DO WHILE statement executes statements in a DO loop repetitively while a condition is true

11.2 Using SAS Functions

SAS functions are built-in routines enabling data manipulations quickly and easily.
Categories of SAS functions include:

- Array
- Bitwise Logical Operations
- Character
- Character String Matching
- Currency Conversion
- Data and Time
- Descriptive Statistics
- Double Byte Character Set
- External Files
- Financial
- Hyperbolic
- Macro
- Mathematical
- Probability
- Quantile
- Random Number
- SAS File I/O
- Special
- State and ZIP Code
- Trigonometric
- Truncation

- Variable Control
- Variable Information
- Web Tools.

With SAS functions:

- Calculate sample statistics
- Create SAS date values
- Convert Zip Codes to state postal codes
- Round values
- Generate random numbers
- Extract a portion of a character value
- Convert data from one type to another.

11.2.1 Basics of SAS Functions

To use a SAS function:

- Format:
  function-name (argument-1 <,argument-n>);
    arguments can be variables, constants, or expressions

If more than one argument, a comma is used to separate arguments. If the argument is a list or array, the argument must be preceded by the word OF to be interpreted correctly.

Target variables are the variables where the result of a function is assigned. The default length of the target variable is dependent on the function, causing more space than necessary in the data set. Add a LENGTH statement for the target variable to control the length of the variable.

11.2.2 Data Conversions

The INPUT function will convert character data values to numeric values. The PUT function will convert numeric data values to character values.

When conversion between variable types is required, SAS will automatically attempt the conversion with unpredictable results. Explicit conversions provide better results.

Automatic character-to-numeric conversion happens when a character value is:

- assigned to a defined numeric variable
- used in an arithmetic operation
- compared to a numeric value
- specified in a function requiring numeric arguments.

Automatic conversions:

- use the w.d informat
- produce a numeric missing value from any character value that does not conform to the standard numeric notation.

WHERE statements do not perform automatic conversions.

The INPUT statement will explicitly convert characters to numeric values. To explicitly convert character to numeric values:

- Format:
  INPUT *(source, informat);*
  - *source* identifies the character variable, constant, or expression to be converted to a numeric value
  - *informat* must be numeric and identified

Automatic numeric-to-character conversion happens when a numeric value is:

- assigned to a defined character variable
- used with an operator that requires character values
- specified in a function requiring character arguments.

The INPUT statement will explicitly convert characters to numeric values.
To explicitly convert numeric to character values:

- Format:
  PUT *(source, format);*
  - *source* identifies the numeric variable, constant, or expression to be converted to a character value
  - *format* must match the data type of the source being specified.

A PUT statement will:

- always return a character string
- return the source written with a format.

The format and the source must agree in type.

11.2.3 Modifying Date Values

Many SAS functions work with data and time values. SAS stores a date value as the number of days from January 1, 1960 to a given date. SAS stores a time value as the number of seconds since midnight. The SAS datetime value is the number of seconds between midnight on January 1, 1960 and a given data and time.

SAS Date Functions

MDY	SAS data (month, day, year)
TODAY DATE	Today's date
TIME	Current time
DAY	Extracts the day value from a SAS data.
QTR	Extracts the QTR value from a SAS data.
WEEKDAY	Represents the days of the week.
MONTH	Extracts the month value from a SAS data.
YEAR	Extracts the year value from a SAS data.
INTCK	Returns the number of time intervals that occur in a given time span.

INTNX	Applies multiples of a given interval to a date, time, or datetime to obtains a result.
DATDIF YRDIF	Calculates the days or years between two SAS dates.

11.2.4 Modifying Character Values

Functions which modify character values can:

- separate character string variables into multiple smaller variables
- replace a portion of a character variable's values
- search for a specific string.

Character functions include:

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, removing leading and trailing blanks and inserts separators.
INDEX	Searches a character value for a specific string.
FIND	Searches for a specific substring of characters within a character string specified.
UPCASE	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 can be used to separate a character value into words and to return a specified word.

Delimiters are used to separate a character string into words. Two or more contiguous delimiters are treated as one delimiter.

Default delimiters include: blanks . < (+ | & ! $ *) ; ^ - / , %

To use a SCAN function with delimiters:

- Format:
  SCAN *(argument, n, delimiters);*
  - *argument* identifies the character variable or expression to be scanned
  - *n* identifies which word to read
  - *delimiters* are special characters enclosed in parentheses.

The SCAN function assigns a length of 200 to every target variable. A LENGTH statement in the DATA step will limit the length.

The SUBSTR function allows:

- a portion of the character value to be extracted
- a portion of the character value to be replaced.

To use a SUBSTR function:

- Format:
  SUBSTR *(argument, position, <n>);*
  - *argument* identifies the character variable or expression to be scanned
  - *position* is the character position to start from
  - *n* identifies the number of characters to extract.

To use a TRIM function to remove trailing blanks:

- Format:
  TRIM *(argument);*
  - *argument* is any character expression: character variable or character function

To use a CATX function for concatenation:

- Format:
  CATX *(separator, string-1 <,...string-n>);*
  - *separator* identifies the character string to be used between concatenated strings
  - *string* specifies a SAS character string.

To use a INDEX function to search for a string within a value:

- Format:
  INDEX *(source, excerpt);*
  - *source* is the character variable or expression to search
  - *excerpt* is the character string enclosed in quotation marks.

To use a FIND function to search for a string within a character string:

- Format:
  FIND *(string, substring<,modifiers>);*
  - *string* is the character constant, variable, or expression to be searched
  - *substring* is the character constant, variable, or expression to be searched for in string
  - *modifiers* is the character constant, variable, or expression which specifies one or more modifiers
  - *startpos* is an integer specifying the position where the search should start and its direction.

To use an UPPERCASE function:

- Format:
  UPCASE *(argument);*
  - *argument* is any character expression: character variable or character constant.

To use a LOWERCASE function:

- Format:
  LOWCASE *(argument);*
  - *argument* is any character expression: character variable or character constant.

To use a PROPCASE function to convert all words to proper names:

- Format:
  PROPCASE *(argument <,delimiter(s)>);*
  - *argument* is any character expression: character variable or character constant
  - *delimiter(s)* identify the separators of the string.

To use a TRANWRD function to replace or remove all occurrences of a pattern:

- Format:
  TRANWRD *(source, target, replacement)*
  - *source* is the string to translate
  - *target* is the string to be searched for in the source
  - *replacement* is the string that replaces target.

11.2.5 Modifying Numeric Values

To return the integer portion of a numeric value:

- Format:
  INT *(argument);*
  - *argument* is any numeric variable, constant or expression.

To round values to the nearest unit:

- Format:
  ROUND *(argument,round-off-unit);*
  - *argument* is any numeric variable, constant or expression
  - *round-off-unit* is numeric and nonnegative.

11.3 Expanding DO Loops

DO loops allow:

- concise DATA steps to be written
- data to be generated
- statements to be conditionally executed
- data to be read.

Groups of statements are processes repeatedly, not just once.

To construct a DO loop:

- Use a DO statement
- Format:
  DO index-variable=start TO stop BY increment;
  SAS statements
  END;
    - index variable will store the value of the current iteration of the DO loop
    - The start, stop, and increment values:
        - are set when the entering the DO loop
        - cannot be changed during processing
        - can be numbers, variables, or SAS expressions.

11.3.1 Executing DO Loops

Initiating DO loops within DATA steps begin with compiling the data. A program data vector is created. When the DATA step is executed, the values are assigned to the variables that will be used within the DO loop.

The DO loop executes where the specified calculations are made and the result added to the previous value for the variable in the program data vector. The DO loop continues to execute for as many times as specified in the program. At this point the DO loop ends and finally the DATA step.

If an observation for each iteration of the DO loop is desired, place an OUTPUT statement inside the loop.

The DO loop's index variable can be decremented by specifying a negative value for the BY value of the DO statement.

The number of times a DO loop is executed can be specified by listing items in a series. The DO loop will execute for each item listed in the series. The items in the series must be separated by commas.

DO loops can be nested in a DO loop. For multiple nested DO loops, a unique index-variable name must be assigned for each iterative DO statement. Each DO loop must have an END statement.

The number of times a DO loop is executed can be conditionally determined using the DO UNTIL and DO WHERE statements.

DO UNTIL statements are used when DO loops should be executed repeatedly until a condition is met.
To execute a DO loop until a condition is met:

- Use a DO UNTIL statement
- Format:
  DO UNTIL *(expression);*
  more SAS statements
  END;

  expression is a valid SAS expression enclosed in parentheses.

The DO UNTIL statement will execute at least once.

DO WHILE statements are used when DO loops should be executed repeatedly while a condition is met.
To execute a DO loop until a condition is met:

- Use a DO WHILE statement
- Format:
  DO WHILE *(expression);*
  more SAS statements
  END;

  expression is a valid SAS expression enclosed in parentheses

If the expression is false during the first iteration, the DO loop will never execute.

DO loops can be used to draw sample observations from a data set. To create sample data, enclose a SET statement in the DO loop. Use the start, stop, and increment values to control which observations to select from.

A POINT=option should be associated with the index variable.

11.4 Process Arrays

Arrays can be used to:

- perform repetitive calculations
- create several variables with the same attributes
- read data
- change variables to observations
- change observations to variables
- compare variables
- perform a table lookup.

A SAS array is a temporary grouping of variables under a single name. An array exists for the duration of the DATA step. Arrays are used to reduce the number of statements required for processing variables.

To group variables into an array:

- Use an ARRAY statement
- Format:
  ARRAY *array-name{dimension} <elements>;*
  - *array-name* specifies the name of the array
  - *dimension* shows the number and arrangement of array elements. The default dimension is one.
  - *elements* list the variables to include in the array.

11.4.1 One-dimensional Arrays

A dimension for a one-dimensional array can be specified in several ways:

- The number of array elements is specified
- A range of values is specified
- An asterisk will have the SAS count the number of elements
- The dimension can be enclosed in parentheses, graces, or brackets.

When elements are specified, they need to be separated by a space. Elements can be specified as a variable list.
Common variable lists:

a numbered range of variables	Var1-Varn
all numeric variables	_NUMERIC_
all character variables	CHARACTER
all variables	_ALL_

The advantage of arrays is the ability to reference the elements by an index value. An index value is assigned to each array element. An array reference is used to perform actions on an array element during execution.

To create an array reference:

- Format:
  array-name{index value}
  index value:
  - is enclosed in parentheses, braces, or brackets
  - specifies an integer, numeric variable, or SAS numeric expression
  - Is within the lower and upper bounds of the array's dimension.

11.4.2 Executing Arrays

During compilation, a program data vector is created. The DATA step is scanned for syntax errors. The index values of the array elements are assigned. Since an ARRAY statement is a compile-time only statement, it is ignored during execution. The DATA step executes.

The DIM function can be used to return the number of elements in the array.

- Format:
  DIM (*array-name)*

11.4.3 Advanced Array Functions

Variables can be created in an ARRAY statement. Omit the array elements from the ARRAY statement. With no array elements, SAS will automatically create variables and assigns default names to them.

To create variables into an array:

- Use an ARRAY statement
- Format:
  ARRAY *array-name{dimension};*
  - *array-name* specifies the name of the array
  - *dimension* shows the number and arrangement of array elements. The default dimension is one.

Default variable names are created by concatenating the array name and the numeric value of the dimension.

An array of character variables can be created by adding a dollar sign after the array dimension.
By default, the length of the variable name is 8, but that can be changed by assigning a length after the dollar sign.

To assign initial values in an ARRAY statement:

- place the values after the array elements
- specify an initial value for each corresponding array element
- separate each value with a blank or comma
- enclose the initial values in parentheses.

Temporary array elements for the DATA step process can be done by specifying _TEMPORARY_ after the array name and dimension. Temporary array elements do not appear in the resulting data set.

11.4.4 Multidimensional Arrays

Variables can be grouped into table-like structures, called multidimensional arrays. To define a multidimensional array, the number of elements in each dimension is specified separated by a comma.

The first dimension specifies the number of rows. The second dimension specifies the number of columns. Array elements are grouped in the order they are listed in the ARRAY statement.

Multidimensional arrays are typically used with nested DO loops.

Arrays can also rotate a SAS data set:

- changing variables into observations
- changing observations into variables.

12 Troubleshooting SAS Programs

SAS programs consist of SAS statements. A consistent layout enhances readability.

Best practices:

- begin DATA and PROC steps in column one
- indent statements within a step
- include a RUN statement after every DATA or PROC step
- begin RUN statements in column one.

12.1 Types of Errors

The most common errors:

- syntax errors – occur when statements do not conform to language rules
- data errors – occur when data values are not appropriate for the specified SAS statements.

12.1.1 Syntax Errors

SAS scans for syntax errors in each step before processing the step. Information, including any errors, is written to the SAS log.

Errors are displayed in the SAS log:

- beginning with the word ERROR
- the possible location of the error
- explanation for the error.

Programs are edited in the Editor window. After correcting programs, you resubmit it.

12.1.2 Resolving Common Problems

Common errors that may require editing:

- spelling mistakes
- omitted semicolons
- unbalanced quotation marks
- invalid options specified.

Many errors come with messages in the SAS log:

- omitted semicolon – log message indicates an error in a statement which seems valid
- unbalanced quotations – log message indicates test string is too long or ambiguous
- invalid options – log message indicated option is invalid or unrecognized.

A missing RUN statement will have an error message “DATA (or PROC) step running” at the top of the active window.

Each step in a SAS program is compiled and executed independent of all other steps. A DATA or PROC statement indicated the beginning of the step. A RUN or QUIT statement indicated the end of the step. To resolve, submit a RUN statement.

13 Practice Exam

The following multiple-choice questions are a refresher from the Foundation level as a prelude.

Question 1

Which of the following items is not considered a category for SAS functions?

A) Special
B) Descriptive Statistics
C) Arithmetic Operations
D) Variable Control

Question 2

Statements are free format expressions that make up the SAS program. Which of the following are true about statements?

A) Statements must begin at the start of the line
B) Statements are always in lowercase to delineate from keywords
C) Only one statement is allowed any line of programming code
D) Statements always end in a semicolon

Question 3

Which of the following is true about priorities in relation to calculating arithmetic expressions?

A) If consecutive operators have the same priority, they are always read from right to left
B) Exponential calculations will always be done before multiplication in any simple expression
C) Multiplication and addition have a higher priority than subtraction
D) Arithmetic operations have to be enclosed by parentheses

Question 4

Which of the following statements are correct when using the RETAIN statement?

A) Variables read with the SET statement will not be affected by the RETAIN statement
B) The RETAIN statement is used when a variable needs to be initialized to a value of 0
C) Variables will be initialized to the desired value each time the DATA step is executed
D) Variables are created during execution

Question 5

When multiple expressions are used within a function, what delimiter should be used to separate them?

A) Space
B) Semicolon
C) Parentheses
D) Comma

Question 6

Which of the following statements is not correct about crosstabulation of data?

A) Frequency calculations for the number of cells that appear will be included in the table
B) Crosstabulation is used when comparing three or more distinct variables
C) The first variable listed in the TABLES statement will define the rows of the table
D) Each level of variables will be represented by a separate two-way table

Question 7

Which of the following statements is not true for list inputs?

A) Fields cannot be skipped or re-read
B) Fields are read from left to right
C) Fields must be separated by a delimiter
D) The MISSOVER option must be used at all times to read missing values

Question 8

Each data set has a descriptive portion providing useful information about the data set. Which of the following items would not be found in the descriptive portion?

A) Creation data and time
B) Set name
C) Information on labels
D) Error messages

Question 9

What would be the correct raw data value of a datatime field for Dec 15, 1986 10:00pm for SAS?

A) 10942
B) 10935
C) 15756360
D) 121519862200

Question 10

Which location pointer identifies the exact line number from which to begin reading data from?

A) #n pointer
B) input pointer
C) / pointer
D) Column pointer

Question 11

What is the purpose of the SUM statement?

A) To combine numeric variables
B) To obtain a total output of multiple values
C) To produce column totals
D) All of the above

Question 12

Which programming statement is used to perform conditional processing on groups of SAS statements?

A) DO
B) BY
C) WHERE
D) IF/THEN

Question 13

Which character function would be used to remove extra spaces from character strings?

A) SUBSTR
B) TRIM
C) FIND
D) CATX

Question 14

Which of the statements below is incorrect?

A) SAS will read and execute steps simultaneously
B) A step is completed when the QUIT statement appears
C) All programming steps begin with either a DATA or PROC keyword
D) A step is completed when the RUN statement appears

Question 15

Which of the following statements is true about column inputs of data?

A) Once a field is read, the data step cannot come back to it
B) Should not be used to read nonstandard numeric data
C) All fields have to be read sequentially
D) Missing data must have a placeholder in the form of a period

Question 16

Which of the following is true about performance testing of DATA steps?

A) Performance testing cannot be done on data steps
B) A data set is created during performance testing
C) The number of observations can be limited using the OBS= in the FILENAME statement
D) Conditions can be placed in the performance testing to enable value testing

Question 17

Which of the following conditions should discourage the use of the one-to-one readings to combine data sets?

A) When a better method is available
B) All observations from each data set read should be contained in the final data set
C) All variables from each data set read should be contained in the final data set
D) Variables have the same name in multiple data sets

Question 18

When is the best time to use the DROP= statement?

A) Only when used with the KEEP= statement
B) Only when a variable should not be processed
C) When the variable should not appear in the data set
D) Never

Question 19

In order to sequentially read multiple records to create a single observation, what condition must be in place to succeed?

A) The records must be in free format.
B) Variables have to be specified explicitly
C) The #n line pointer has to be specified
D) The number of observations for each record must be the same

Question 20

When would you use a CLASS statement?

A) The CLASS statement is used to sort data in a defined structure
B) The CLASS statement is used to define and group observations
C) The CLASS statement is used when applying a specified statistic to a group of observations
D) The CLASS statement is used when a table must be created

Question 21

Which of the following is not considered a character type of variable?

A) Miller
B) Miller12
C) Miller.2
D) 2miller4u

Question 22

When would the keyword, FILE, be appropriate to use?

A) When attempting to describe the file where raw data is being written
B) When referencing the file to be read
C) When creating a raw data file
D) When referencing the file to be written to

Question 23

What data summarizations are provided by the PROC MEANS statement by default?

A) The minimum value
B) the n-count
C) The standard deviation
D) All of the above

Question 24

Which of the following is a true statement about executing a data step?

A) The input raw data is read from the input buffer
B) Error checking is performed at the beginning of the execution phase
C) Multiple observations can be found in a single fixed field record
D) The iteration for a DATA step refers to the number of INPUT statement in the step

Question 25

When using the VALUE statement to define the format of specific values, which of the following statements is not correct about identifying the range.

A) A series of character values can be identified as the range for a VALUE statement
B) A single value can be identified as the range for a VALUE statement
C) A list of unique values separated by commas can be identified as the range for a VALUE statement
D) A series of numeric values can be identified as the range for a VALUE statement

Question 26

Which of the following describes the programming process of adding observations from one data set to the observations of another data set?

A) Match-merging
B) Interweaving
C) Concatenating
D) None of the above

Question 27

What statement is used to identify the conditions for selecting observations in a report?

A) WHERE
B) IF/THEN ELSE
C) SET
D) INPUT

Question 28

Which of the following items can be expected by the SORT procedure?

A) Replace the original SAS data set
B) Will not generate printed output
C) Rearrange observations in the SAS data set
D) All of the above

Question 29

When is character-to-numeric conversion not automatically performed?

A) When used in an arithmetic operation
B) When used within a WHERE statement
C) When compared to a numeric value
D) When assigned a defined numeric variable

Question 30

Several libraries are created during a SAS session by default. Which of the following is not one of those libraries?

A) Sasuser
B) Libname
C) Sashelp
D) Work

Question 31

When are SAS arrays not used?

A) SAS arrays are used to compare variables
B) SAS arrays are used to change variables to observations
C) SAS arrays are used to group data within the data set
D) SAS arrays are used to perform repetitive calculations

Question 32

Which of the following in a correctly created data set name?

A) _Set.name
B) addAnew_1
C) 0pen_2C1other_rtm*XCX
D) create\B4infinity

Question 33

What must happen for DO Loops to be nested within each other?

A) Each DO loop must end in a RUN statement
B) A condition must be in place to determine if the DO loop should be done
C) An OUTPUT statement is required to ensure the result of the DO-loop is retained
D) A unique index-variable name must be assigned for each iterative DO statement

Question 34

Which of the following programming conventions can be used in conjunction with the line-hold specifier, @@?

A) @ pointer control
B) MISSOVER option
C) column input
D) INPUT statement

Question 35

Which of the following is not an error identified during the compilation phase?

A) Quotation marks are unbalances
B) No RUN statement in the step
C) An option is invalid
D) Semicolons are missing in statements

Question 36

Which of the following methods for combining data sets is distinguished for its use of the BY statement?

A) Interweaving
B) Match-merging
C) Concatenation
D) None of the above

Question 37

Which of the following options does not describe how variables are to be used in creating a report?

A) DISPLAY
B) ACROSS
C) GROUP
D) NUMERIC

Question 38

Which of the following is a standard numeric data value?

A) Integer binaries
B) Scientific notation
C) Fractions
D) Hexadecimal numbers

Question 39

Which of the following keywords is used to create a raw data file?

A) filename
B) libname
C) infile
D) set

Question 40

Which of the following is not an attribute of the program data vector?

A) Will scan data for errors
B) Is created after the creation of the input buffer
C) Refers to physical and logical memory available to the SAS program
D) Will expand as the number of variables increase

14 Answer Guide

14.1 Answers to Questions

Question 1
Answer: C
Reasoning: There are many categories for SAS functions, but Arithmetic Operations is not one of them. The category is actually Bitwise Logical Operations.

Question 2
Answer: D
Reasoning: Statements can start anywhere in a line which also allows for multiple statement to exist on a single line. There are no restrictions to the case required for statements. Statements always begin with keywords and end in a semicolon.

Question 3
Answer: B
Reasoning: Exponential expressions have a higher priority than multiplication expressions, unless parentheses exist to control any calculations.

Question 4
Answer: A
Reasoning: The RETAIN statement is used when a variable needs to be initialized to a value other than 0. The statement will prevent variables from being initialized. All of this is done during the compilation stage, not the execution stage. Variables that are read with the SET, MERGE, and UPDATE statements are not affected by the RETAIN statement.

Question 5
Answer: D
Reasoning: Multiple variables, constants, and expressions must be separated by a comma when used in an argument.

Question 6
Answer: B
Reasoning: Crosstabulation is used when comparing two distinct variables. More variables may be compared as well.

Question 7
Answer: D
Reasoning: If missing values are represented by a special character that is not used as a separated of data, then the MISSOVER option is not required.

Question 8
Answer: D
Reasoning: Error messages are found in the program log, not the descriptive portion of the data set.

Question 9
Answer: C
Reasoning: The datetime value is the number of seconds from midnight on 1/1/1960. This of course would be a rather large number, in fact, 120 seconds from 10942 days (including 7 extra days from leap years). The last answer is incorrect as it is the date and time written out without any delimiters.

Question 10
Answer: A
Reasoning: The #n pointer references the absolute number of the line for moving the pointer to read data.

Question 11
Answer: C
Reasoning: In creating reports for SAS programs, the SUM statement is used to create column totals for numeric values.

Question 12
Answer: A
Reasoning: The key word in this question is ‘group’.   Conditional processing can be performed using some iteration of BY, WHERE, and IF/THEN, but DO provides an opportunity to group statements together.

Question 13
Answer: D
Reasoning: CATX is the most correct answer as it removes both leading and trailing blanks from character strings. TRIM only removes the trailing blanks from character values.

Question 14
Answer: A
Reasoning: Programming steps begin with DATA or PROC and end with either QUIT or END. SAS steps are completely read before executing.

Question 15
Answer: B
Reasoning: Column input is used when data fields are fixed. In this situation, the columns can be read in any order and re-read if necessary. Missing data does not require a placeholder either. Unfortunately, only standard data should be read using column input: nonstandard data should utilize the formatted input.

Question 16
Answer: D
Reasoning: Performance testing can be done on data steps by putting a NULL in the DATA step. This will prevent a data set from being created. The number of observations can be limited by putting the OBS= option in the INFILE statement. Conditions can be set using IF/THEN ELSE statements to test for values.

Question 17
Answer: B
Reasoning: One-to-one readings are the simplest form of combining data sets which will provide the best option in ensuring the programming code is not overly complex. In this type of combining, all the variables from each of the data sets will exist in the new data set. Variables with the same name does not matter, since the values of the last data set read will overwrite all previous values from existing data sets. If the values should not be overwritten, the variable name can be changed before the output is rendered. Not all observations will appear in the final data set: if this is a requirement than concatenation is the next best choice for combining data sets.

Question 18
Answer: C
Reasoning: DROP= and KEEP= can be used independently from each other. Whether the DROP= statement appears in the same statement as the DATA or SET statement will determine if the variable will be processed or not. In all cases though, the variable will not appear in the data set.

Question 19
Answer: D
Reasoning: The only correct answer is that the number of observations for each record must be the same.

Question 20
Answer: C
Reasoning: The CLASS statement will allow statistics to be applied to a group of observations. As a result of this statement, the data would be sorted, grouped and displayed in a single table; but is not the primary reason for using the CLASS statement.

Question 21
Answer: C
Reasoning: SAS naming conventions require variables names to be 1-32 characters long, can begin and consist of any combination of letters, numbers, and underscores.

Question 22
Answer: D
Reasoning: SET is used to create a raw data file. FILENAME references a file to be read, while FILE is used to reference a file to be written to. The PUT statement will describe the lines written to the raw data.

Question 23
Answer: D
Reasoning: PROC MEANS will also provide the mean and the maximum value without limiting the output.

Question 24
Answer: B
Reasoning: Input raw data is read from the program data vector while INPUT statements are read from the input buffer. In raw data with fixed fields, a single record provides only one observation. Multiple observations can be contained in a single record for free-format files. An iteration refers to the complete cycle through the data step, typically one iteration for each observation found.

Question 25
Answer: A
Reasoning: C shows that detail is required in the answer. The range can consist of character values which are enclosed by quotation marks.

Question 26
Answer: C
Reasoning: Match merging will merge several data sets together based on the value of a common variable. Interweaving can do the same as match-weaving, but with more than one common variable. Concatenation will append, or add on, to existing data sets.
.
Question 27
Answer: A
Reasoning: The WHERE statement will identify conditions for identifying variables and values to print in a report.

Question 28
Answer: D
Reasoning: All of the situations describe are possible with the SORT procedure.

Question 29
Answer: B
Reasoning: A WHERE statement will not use automatic conversion of any kind.

Question 30
Answer: B
Reasoning: All of the libraries are created by default except libref, which is not a library but a keyword for naming a library.

Question 31
Answer: C
Reasoning: SAS arrays are a temporary grouping of variables that exist for the duration of a data step. They will not 'group' data within the data set.

Question 32
Answer: B
Reasoning: Data set names can be 1-32 characters long and consists and begin with alphanumeric characters and underscores.

Question 33
Answer: D
Reasoning: DO statements must end with an END statement. A condition is not necessary. The OUTPUT statement is used to create observations for each iteration of the DO loop. A unique index-variable name is required when nesting DO loops.

Question 34
Answer: D
Reasoning: A, B, C should not be used with the double training at line-hold specifier.

Question 35
Answer: B
Reasoning: The compilation phase will identify most syntax errors, while the executing phase will identify processing errors. No RUN or QUIT statement will result in processing to continue indefinitely.

Question 36
Answer: A
Reasoning: Match-merging is distinguished by the MERGE statement and concatenation by multiple SET statements. The interweaving method uses BY statements to choose which observations are appropriate from each of the data sets. Though the match-merging method also uses BY statements to focus the observations used, it must have the MERGE statement to perform properly.

Question 37
Answer: D
Reasoning: There is no numeric options: the default for numeric values is ANALYSIS. Additional options not mentioned include ORDER and COMPUTED.

Question 38
Answer: B
Reasoning: Standard numeric data values consist of positive and negative numbers, decimal points, and scientific notation.

Question 39
Answer: D
Reasoning: 'Libname' is used to create a library. 'Filename' will reference an external file where raw data is stored and after it has been referenced once, the 'infile' is used to identify the external file. 'Set' will create a raw data file.

Question 40
Answer: C
Reasoning: The program data vector refers to logical memory only.

15 References

SAS Certification Prep Guide: Base Programming for SAS 9, www.scripd.com

For more information:
www.support.sas.com

Websites

www.artofservice.com.au
www.theartofservice.org
www.theartofservice.com

INDEX*