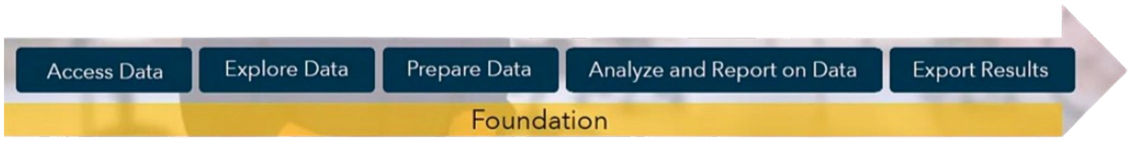
1.sasGetStart

March 3, 2025

0.0.1 Getting Started with SAS

Before you start writing SAS programs you need to know the essentials. Wich are SAS programming tools and the fundamentals of SAS program structure and syntax.



THE SYNTAX :

Module Summary: Essentials

Running a SAS Program

- Programs can be submitted by clicking the **Run** icon or pressing the F3 key.
- To run a subset of a program, highlight the desired statements first. If you are using SAS Studio, click the **Run** icon or press F3. If you are using SAS Enterprise Guide, click the arrow next to **Run** and click **Run Selection** or press F3.
- A program can create a log, results, and output data.
- Submitting a program that has run previously in Enterprise Guide or SAS Studio replaces the log, output data, and results.
- Submitting a program that has run previously in the SAS windowing environment appends the log and results.

Understanding SAS Syntax

- SAS programs consist of DATA and PROC steps, and each step consists of statements.

```
DATA ... ;
      other statements
RUN;
```

```
PROC ... ;
      other statements
RUN;
```

- Global statements are outside steps.

```
TITLE ... ;
```

```
OPTIONS ... ;
```

```
LIBNAME ... ;
```

- All statements end with a semicolon.
- Spacing doesn't matter in a SAS program.
- Unquoted values can be lowercase, upper case, or mixed case.

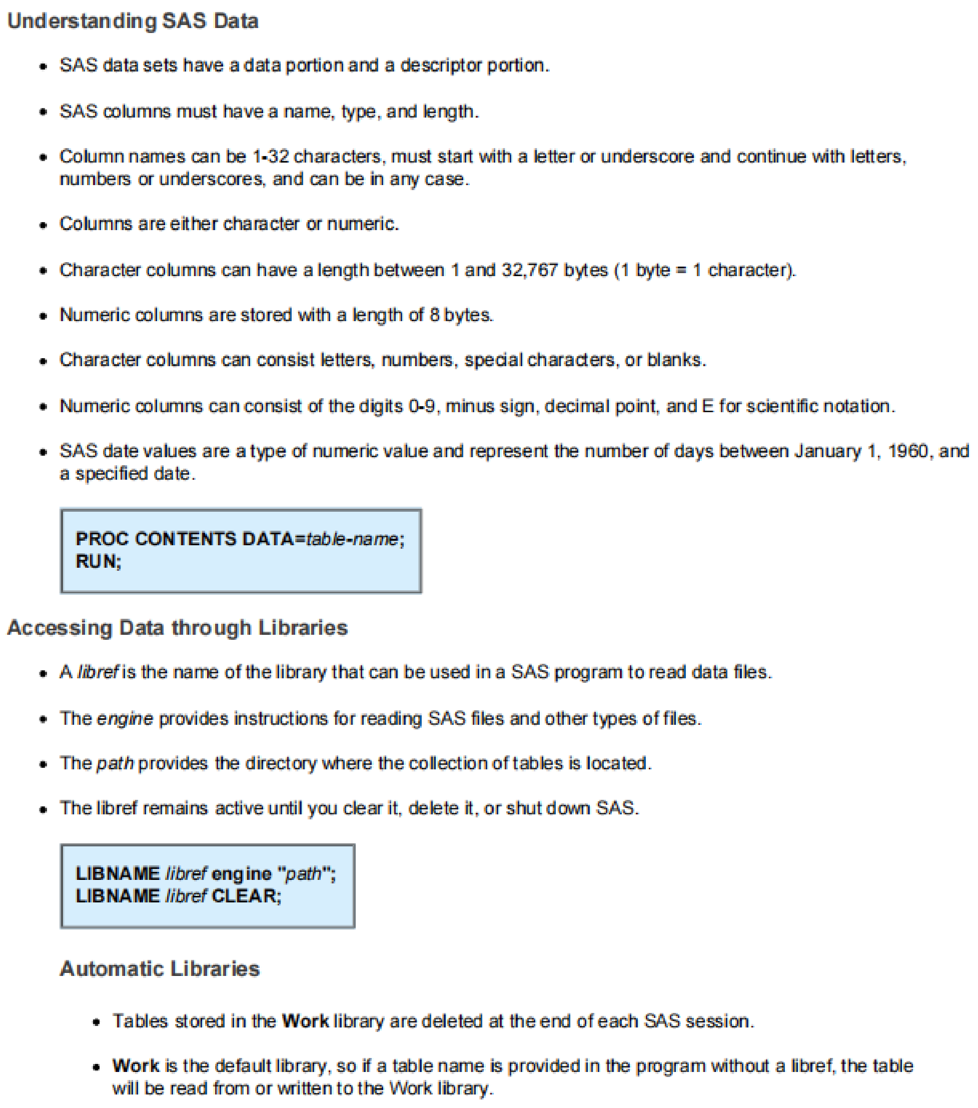
- Consistent program spacing is a good practice to make programs legible.
- Use the following automatic spacing features:

  SAS Studio: Click the **Format Code** icon.
  Enterprise Guide: Select **Edit** > **Format Code** or press Ctrl+I.

- Comments can be added to prevent text in the program from executing.
- Some common syntax errors are unmatched quotes, missing semicolons, misspelled keywords, and invalid options.
- Syntax errors might result in a warning or error in the log.
- Refer to the log to help diagnose and resolve syntax errors.

0.0.2 Accessing Data

In Accessing Data, you learn to identify the features of a SAS table, access data through libraries, and import data into SAS. (Data can come from a variety of locations and in many formats. Your data might be structured or unstructured. It might be SAS data, a Microsoft Excel file....)



Understanding SAS Data

- SAS data sets have a data portion and a descriptor portion.
- SAS columns must have a name, type, and length.
- Column names can be 1-32 characters, must start with a letter or underscore and continue with letters, numbers or underscores, and can be in any case.
- Columns are either character or numeric.
- Character columns can have a length between 1 and 32,767 bytes (1 byte = 1 character).
- Numeric columns are stored with a length of 8 bytes.
- Character columns can consist letters, numbers, special characters, or blanks.
- Numeric columns can consist of the digits 0-9, minus sign, decimal point, and E for scientific notation.
- SAS date values are a type of numeric value and represent the number of days between January 1, 1960, and a specified date.

```
PROC CONTENTS DATA=table-name;
RUN;
```

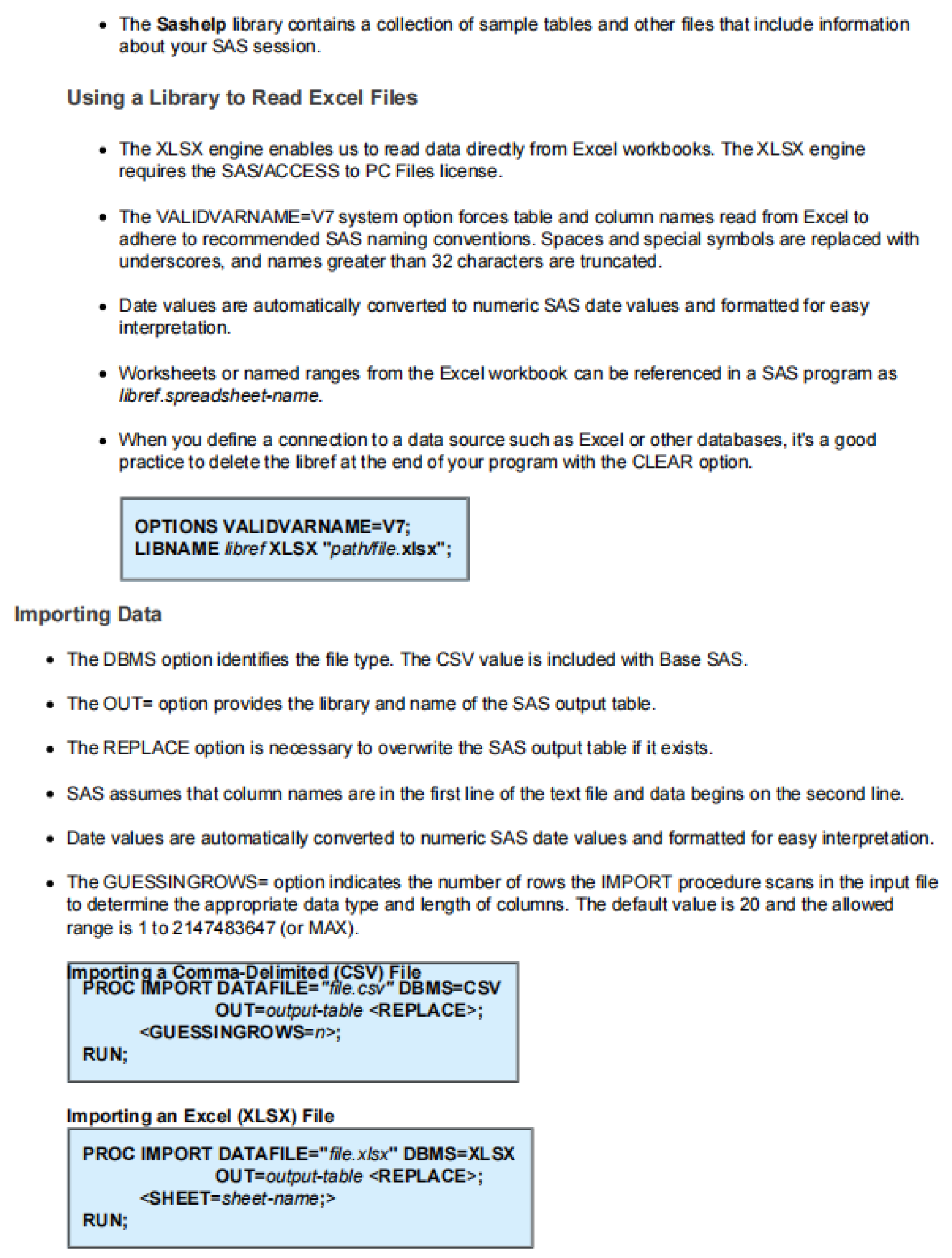
Accessing Data through Libraries

- A *libref* is the name of the library that can be used in a SAS program to read data files.
- The *engine* provides instructions for reading SAS files and other types of files.
- The *path* provides the directory where the collection of tables is located.
- The libref remains active until you clear it, delete it, or shut down SAS.

```
LIBNAME libref engine "path";
LIBNAME libref CLEAR;
```

Automatic Libraries

- Tables stored in the **Work** library are deleted at the end of each SAS session.
- **Work** is the default library, so if a table name is provided in the program without a libref, the table will be read from or written to the Work library.



- The **Sashelp** library contains a collection of sample tables and other files that include information about your SAS session.

Using a Library to Read Excel Files

- The XLSX engine enables us to read data directly from Excel workbooks. The XLSX engine requires the SAS/ACCESS to PC Files license.
- The VALIDVARNAME=V7 system option forces table and column names read from Excel to adhere to recommended SAS naming conventions. Spaces and special symbols are replaced with underscores, and names greater than 32 characters are truncated.
- Date values are automatically converted to numeric SAS date values and formatted for easy interpretation.
- Worksheets or named ranges from the Excel workbook can be referenced in a SAS program as *libref.spreadsheet-name*.
- When you define a connection to a data source such as Excel or other databases, it's a good practice to delete the libref at the end of your program with the CLEAR option.

```
OPTIONS VALIDVARNAME=V7;
LIBNAME libref XLSX "path/file.xlsx";
```

Importing Data

- The DBMS option identifies the file type. The CSV value is included with Base SAS.
- The OUT= option provides the library and name of the SAS output table.
- The REPLACE option is necessary to overwrite the SAS output table if it exists.
- SAS assumes that column names are in the first line of the text file and data begins on the second line.
- Date values are automatically converted to numeric SAS date values and formatted for easy interpretation.
- The GUESSINGROWS= option indicates the number of rows the IMPORT procedure scans in the input file to determine the appropriate data type and length of columns. The default value is 20 and the allowed range is 1 to 2147483647 (or MAX).

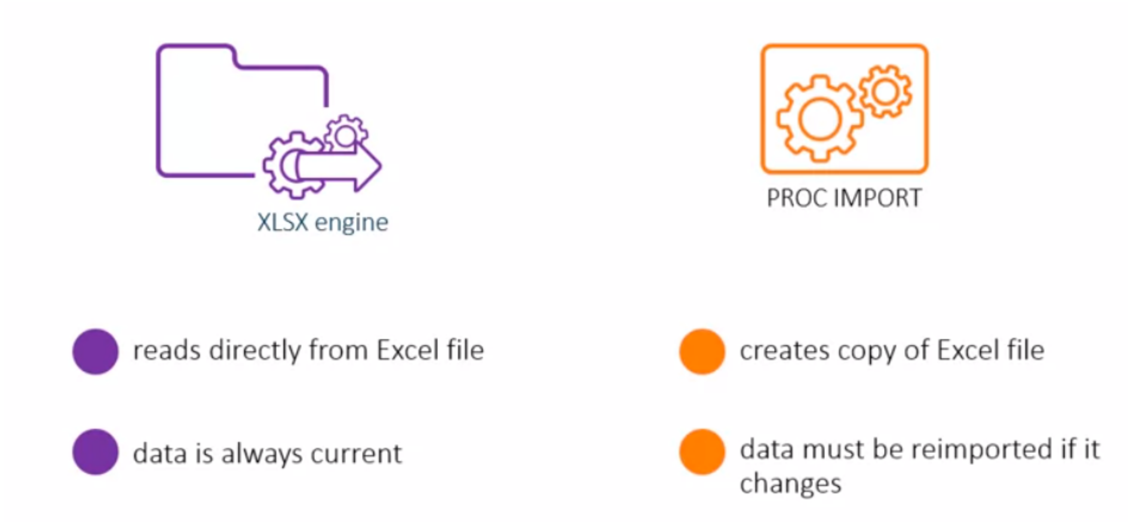
Importing a Comma-Delimited (CSV) File

```
PROC IMPORT DATAFILE="file.csv" DBMS=CSV
            OUT=output-table <REPLACE>;
    <GUESSINGROWS=n>;
RUN;
```

Importing an Excel (XLSX) File

```
PROC IMPORT DATAFILE="file.xlsx" DBMS=XLSX
            OUT=output-table <REPLACE>;
    <SHEET=sheet-name;>
RUN;
```

the diffrence between the import and the libname when it comes to excel files



XLSX engine

- reads directly from Excel file
- data is always current

PROC IMPORT

- creates copy of Excel file
- data must be reimported if it changes

0.0.3 Exploring Data

In Exploring and Validating Data, you learn to use SAS procedures that provide insights about your data. You also learn to subset data so you can focus on particular segments, format data so you can easily understand it, and sort data to identify and resolve duplicate values.

Exploring Data

- PROC PRINT lists all columns and rows in the input table by default. The OBS= data set option limits the number of rows listed. The VAR statement limits and orders the columns listed.

```
PROC PRINT DATA=input-table(OBS=n);
    VAR col-name(s);
RUN;
```

- PROC MEANS generates simple summary statistics for each numeric column in the input data by default. The VAR statement limits the variables to analyze.

```
PROC MEANS DATA=input-table;
    VAR col-name(s);
RUN;
```

- PROC UNIVARIATE also generates summary statistics for each numeric column in the data by default, but includes more detailed statistics related to distribution and extreme values. The VAR statement limits the variables to analyze.

```
PROC UNIVARIATE DATA=input-table;
    VAR col-name(s);
RUN;
```

- PROC FREQ creates a frequency table for each variable in the input table by default. You can limit the variables analyzed by using the TABLES statement.

```
PROC FREQ DATA=input-table;
    TABLES col-name(s) < / options>;
RUN;
```

Filtering Rows

- The WHERE statement is used to filter rows. If the expression is true, rows are read. If the expression is false, they are not.
- Character values are case sensitive and must be in quotation marks.
- Numeric values are not in quotation marks and must only include digits, decimal points, and negative signs.
- Compound conditions can be created with AND or OR.

- The logic of an operator can be reversed with the NOT keyword.
- When an expression includes a fixed date value, use the SAS date constant syntax: "ddmmmyyyy"d, where *dd* represents a 1- or 2-digit day, *mmm* represents a 3-letter month in any case, and *yyyy* represents a 2- or 4-digit year.

```
PROC procedure-name ... ;
     WHERE expression;
RUN;
```

WHERE Operators

= or EQ
^= or ~= or NE
> or GT
< or LT
>= or GE
<= or LE

SAS Date Constant

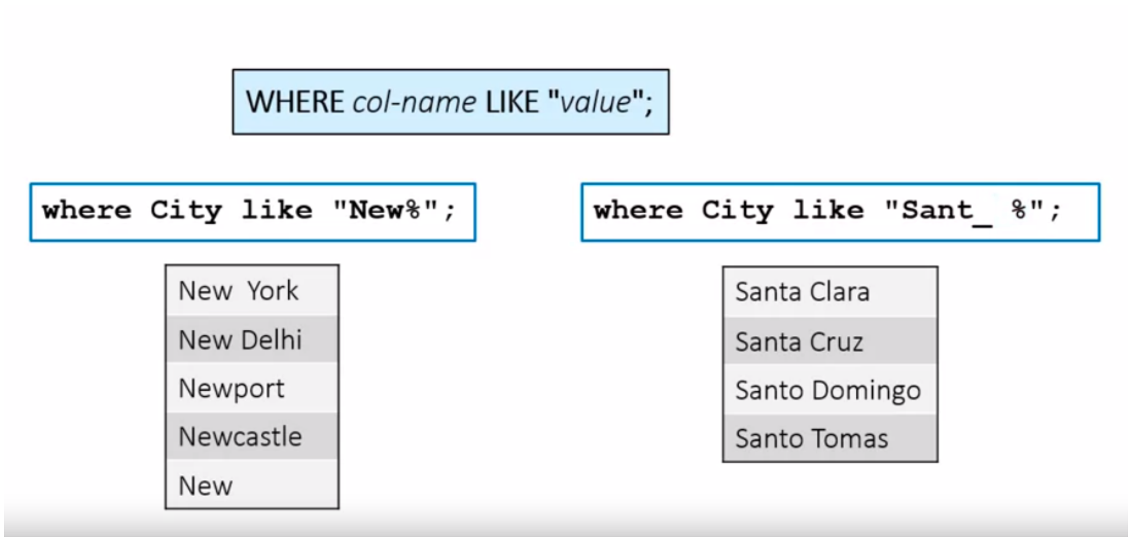
"ddMONyyyy"d

IN Operator

WHERE *col-name* **IN(***value-1<...,value-n>***);**
WHERE *col-name* **NOT IN (***value-1<...,value-n>***);**

Special WHERE Operators

WHERE *col-name* **IS MISSING;**
WHERE *col-name* **IS NOT MISSING;**
WHERE *col-name* **IS NULL;**
WHERE *col-name* **BETWEEN** *value-1* **AND** *value-2*;
WHERE *col-name* **LIKE** "*value*";
WHERE *col-name* **=*** "*value*";



WHERE *col-name* LIKE "*value*";

```
where City like "New%";
```

New York
New Delhi
Newport
Newcastle
New

```
where City like "Sant_ %";
```

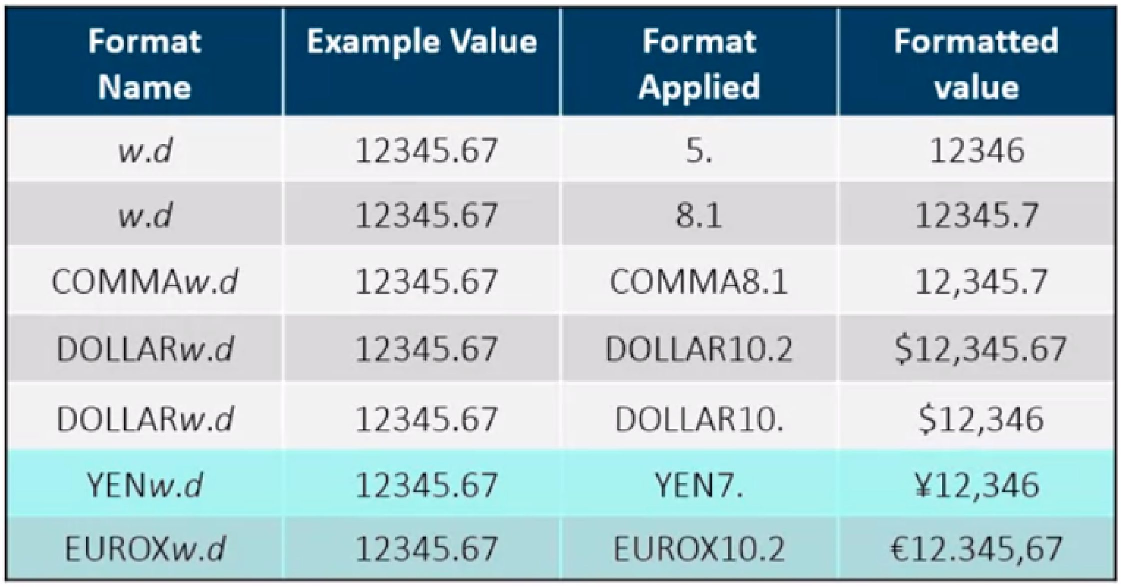
Santa Clara
Santa Cruz
Santo Domingo
Santo Tomas

Formatting Columns

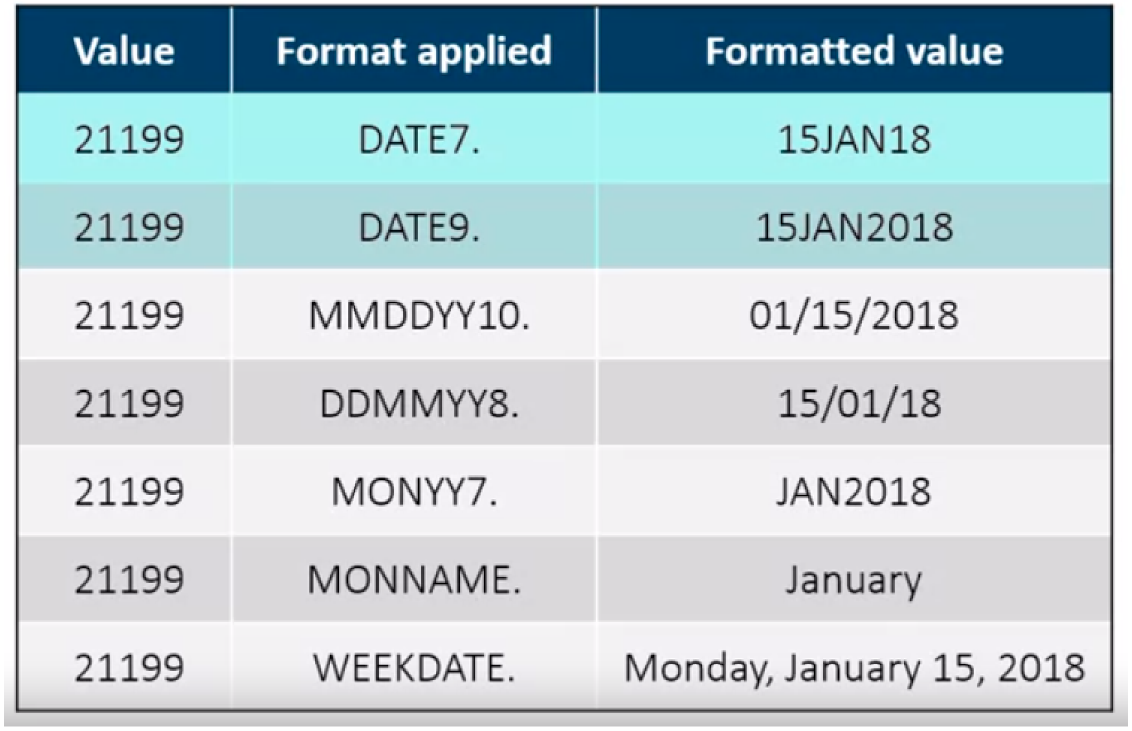
- Formats are used to change the way values are displayed in data and reports.
- Formats do not change the underlying data values.
- Formats can be applied in a procedure using the FORMAT statement.
- Visit SAS Language Elements documentation to access a list of available SAS formats.

```
PROC PRINT DATA=input-table;
      FORMAT col-name(s) format;
RUN;
```

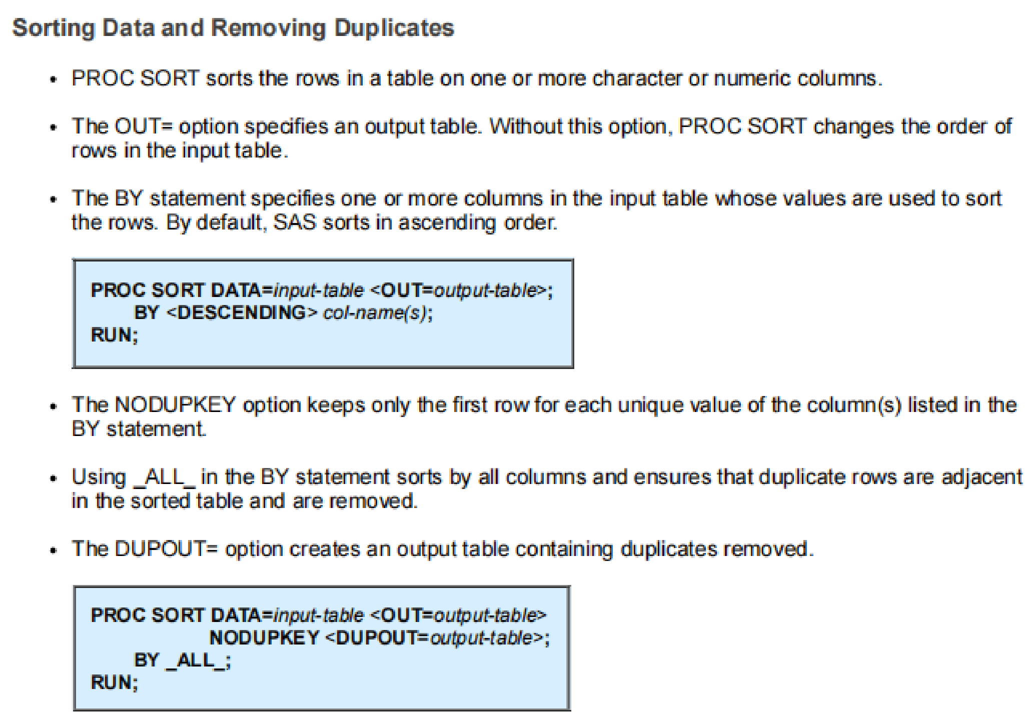
```
<$>format-name<w>.<d>
```



Format Name	Example Value	Format Applied	Formatted value
w.d	12345.67	5.	12346
w.d	12345.67	8.1	12345.7
COMMA*w.d*	12345.67	COMMA8.1	12,345.7
DOLLAR*w.d*	12345.67	DOLLAR10.2	$12,345.67
DOLLAR*w.d*	12345.67	DOLLAR10.	$12,346
YEN*w.d*	12345.67	YEN7.	¥12,346
EUROX*w.d*	12345.67	EUROX10.2	€12.345,67



Value	Format applied	Formatted value
21199	DATE7.	15JAN18
21199	DATE9.	15JAN2018
21199	MMDDYY10.	01/15/2018
21199	DDMMYY8.	15/01/18
21199	MONYY7.	JAN2018
21199	MONNAME.	January
21199	WEEKDATE.	Monday, January 15, 2018



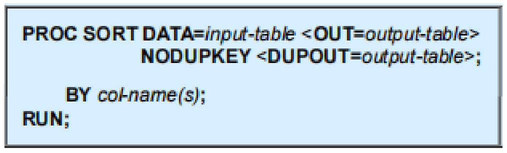
Sorting Data and Removing Duplicates

- PROC SORT sorts the rows in a table on one or more character or numeric columns.
- The OUT= option specifies an output table. Without this option, PROC SORT changes the order of rows in the input table.
- The BY statement specifies one or more columns in the input table whose values are used to sort the rows. By default, SAS sorts in ascending order.

```
PROC SORT DATA=input-table <OUT=output-table>;
      BY <DESCENDING> col-name(s);
RUN;
```

- The NODUPKEY option keeps only the first row for each unique value of the column(s) listed in the BY statement.
- Using _ALL_ in the BY statement sorts by all columns and ensures that duplicate rows are adjacent in the sorted table and are removed.
- The DUPOUT= option creates an output table containing duplicates removed.

```
PROC SORT DATA=input-table <OUT=output-table>
                NODUPKEY <DUPOUT=output-table>;
      BY _ALL_;
RUN;
```



```
PROC SORT DATA=input-table <OUT=output-table>
                NODUPKEY <DUPOUT=output-table>;

      BY col-name(s);
RUN;
```

0.0.4 Preparing Data

```
[ ]:
```