SAS Base Programming Reference Sheet

SAS Programming Fundamentals

- A program can create a log, results, and output data
- Programs are comprised of **Data** and **Procedure** steps
- Steps end with a **run;** statement (sometimes **quit;**)
- Each step is a series a statements
- A statement begins with a keyword (e.g. **data**) and end with a semicolon "**;**"
- Assignment statements do not begin with a keyword

Example Program	Program Explanation
Data myclass; **set** sashelp.class; heightcm = height*2.54; **Run;** **Proc print data =** myclass; **var** age heightcm; **Run;**	Creates a new dataset **myclass** in the work library. Calculates a new variable heightcm Prints a view of the **myclass** dataset in the results window displaying only 2 variables: age, heightcm. There are 7 statements.

Global Statements

- **OPTIONS** *...;* sets SAS system options
  EX: **Options** Validvarname = V7;
- **TITLE** *<options> "...title text...";* set up to 10 titles
  EX: **TITLE** "Student Ages and Heights";
  TITLE2 just= l "Classroom 105";
- **FOOTNOTE** *<options> ".....";* sets up to 10 footnotes
  EX: **FOOTNOTE** "Data refreshed annually";
- **LIBNAME** *libref <engine> ".....";* sets a shortcut reference to data of a specific type in a specific location
  - *libref*: name to call a library. 8-character max length
  - *engine*: contains predefined set of rules for reading data. Base is the default engine reading SAS datasets
  - "....." physical name of the library recognized by the system

  EX: **LIBNAME** mylib "/C:/documents/project/data";
  LIBNAME myXL xlsx "/C:/documents/class.xlsx";
- **LIBNAME** *libref* **CLEAR** ends connection to the data source

Commenting Code

- Comments can be added to prevent text in the program from executing
- There are 2 comment styles
- Comments are not executable statements

```
/* insert commented text here */
* Insert commented text here ;
```

Accessing Data

- SAS can read and understand structured (e.g. xlsx) and unstructured (e.g. csv) data types
- Structured data can be read via a **LIBNAME** statement or **PROC IMPORT** step
- Unstructured data requires **PROC IMPORT** to define rules

Importing data using Proc Import

Example Program	Program Explanation
Proc import datafile = "myfile. csv" **dbms = csv out =** mylib.example **replace;** **guessingrows = 100;** **Run;**	Import a CSV file (unstructured data) that outputs a SAS dataset *example* in a user defined permanent library *mylib*. Optionally add a guessingrows statement to read *n* rows to determine variable attributes.

Procedures for Exploring and Analyzing Data

- **PROC CONTENTS**: Prints descriptor portion of a data set

```
PROC CONTENTS DATA = data-set-name;
RUN;
```

- **PROC PRINT**: lists all columns and rows in the input table by default.
  - **OBS =** option limits the number of rows listed
  - **VAR** statement limits and orders columns listed
    - Use _NUMERIC_, _CHARACTER_, and _ALL_ keywords to specify variables of a certain type (numeric or character) or all types
  - **WHERE** statement filter the data
  - **BY** statement groups output data
  - **FORMAT** statement applies a temporary* format in the output
  - **LABEL** statement applies a temporary* label to the variable names in the output

  ** Permanent characteristics are defined in the data step*

```
PROC PRINT DATA = data-set-name <label> (OBS = n);
    <VAR col-name(s);>
    <WHERE expression;>
    <BY col-names(s);>
    <FORMAT col-name(s) <$> format name. ;>
    <LABEL col-name = "Label"; >
RUN;
```

- **PROC MEANS**: generates simple summary statistics for each numeric column in the input data by default unless the VAR statement is used
  - **CLASS** specifies variables to group data before calculating statistics
  - **WAYS** specifies number of ways to make unique combinations of class variables
  - **OUTPUT** provides the option to create an output table and specific output statistics
  - **OUT =** names the output table to be created

```
PROC MEANS DATA = data-set-name;
    <WHERE expression;>
    <VAR col-name(s);>
    <CLASS col-names(s);>
    <WAYS n;>
    <OUTPUT OUT = output-table <statistic =col-name>
RUN;
```

- **PROC UNIVARIATE**: Generates summary statistics and more detailed statistics about distribution and extreme values for each numeric variable by default

```
PROC UNIVARIATE DATA = data-set-name;
    <VAR col-name(s);>
    <WHERE expression;>
RUN;
```

- **PROC FREQ**: Creates a frequency table for each variable in the input table by default.
  - **TABLES** limits the variables analyzed
  - *<options>* customize outputs by limiting columns (i.e. nocum), modifying output style (i.e. crosslist, listing) or generating graphs.
  - Create a crosstabulation report by adding an asterisk (*****) between two variable names on the **TABLES** Statement

```
PROC FREQ DATA = data-set-name;
    <TABLES col-name(s) </options>;>
    <WHERE expression; >
RUN;
```

SAS Base Programming Reference Sheet

Procedures for Data Manipulation

- **PROC SORT**: sorts the rows in a table on one or more character or numeric columns. A PROC SORT is required before any step that uses a BY statement
  - **BY** specifies the columns used in the sort
  - **OUT =** specifies an output table
  - **NODUPKEY** keeps only the first row for each unique value of the columns(s) listed in the by statement
  - **DUPOUT** creates an output table containing duplicates
  - **DESCENDING** sorts column from 9 to 0 or Z to A

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

- **PROC TRANSPOSE**: is used to restructure a table
  - **VAR** lists column(s) to be transposed
  - **ID** creates a separate column for each value of the ID Variable and can only be one column.
  - **BY** transpose data within groups. Unique combinations of BY values creates one row in the output table
  - **PREFIX** provides a prefix for each value of the ID column
  - **NAME** names the column that identifies the source

```
PROC TRANSPOSE DATA = input-table OUT = output-table  <PREFIX = column> <NAME = column>;
        <BY  col-name(s); >
        <ID column;>
        <VAR columns(s);>
RUN;
```

Preparing Data: The DATA Step

```
DATA output-dataset;
      set input-dataset;
Run;
```

- The data step is processed in two phases:
  - Compilation: creates the PDV, establishes data attributed and rules for execution
  - Execution: SAS reads, manipulates and write data
- DATA Steps create two default variables:
  - **_N_** counts the number of iterations through the data step when processing
  - **_ERROR_** is initialized at 0. If an error is encountered, the value is set to 1
- Explicit Output statements can be used to control when and where each row is written.
- Multiple datasets can be created in one data step

```
DATA work.cheap  work.expensive;
      set work.shopping;
      if price >100 then output work.expensive;
          else output work.cheap
Run;
```

Program Explanation

The data step creates to output tables CHEAP and EXPENSIVE based on the input dataset WORK.SHOPPING. If an observation has PRICE greater than 100, then the observation is assigned to the EXPENSIVE dataset.

The DATA Step: Controlling Variable Output

- **DROP= / KEEP =** options can be added to a table on the DATA statement or SET statement.
- **DROP/ KEEP** statements can be added within the data step
- Columns kept or dropped will be flagged in the PDV
- Dropping a column on the SET statement makes a column unavailable for processing in the data step

```
DATA work.expensive (KEEP = price item_name);
      SET work.shopping (DROP = city state);
      KEEP store_name;
RUN;
```

The DATA Step: Processing Data in Groups

- Process data in groups after sorting data first
- **First.*bycol*** is 1 for the first row within a group and 0 otherwise.
- **Last.*bycol*** is 1 for the last row within a group and 0 otherwise

```
BY col-names(s);
FIRST.bycol <expression>;
LAST.bycol<expression>;
```

- Accumulating columns require modifying SAS' default behavior to retain all PDV values with each iteration.
- Accumulating columns are often used in conjunction with FIRST./LAST. logic allowing BY GROUP calculations & totals
- *Column* is the new variable holding the accumulating total

```
Column + expression;
```

The DATA Step: Conditional Processing and Loops

- Conditionally process data using **IF/ELSE IF/ ELSE** statements
- SAS will check the expressions sequentially until one is true
- **IF** statements can create new variables or new data sets

```
if price >100 then newVar = "Expensive";
else if price <100 and price >0 then newVAR  = "Cheap"
else if price = 0 then newVAR = "FREE";
else newVAR = "Priceless";
```

- Execute multiple statements by using a **DO** statement

```
if price >100 then do;
          newVar = "Expensive";
          output work.expensive;
end;
```

- Process repetitive code using DO LOOPS
- The optional OUTPUT statement will output a row for each iteration of the loop

```
DATA output-table;
      SET input-table;
      DO indexcolumn = start TO stop <BY increment> ;
         . . . repetitive code . . .
          <OUTPUT;>
      END;
RUN;
```

- A DO UNTIL executes until a condition is true, and the condition is checked at the ***bottom*** of the DO loop. A DO UNTIL loop always executes at least one time.
- A DO WHILE executes while a condition is true, and the condition is checked at the ***top*** of the DO loop. A DO WHILE loop does not iterate even once if the condition is initially false.

```
DO  WHILE | UNTIL expression;
   . . . repetitive code . . .
  END;
```

SAS Base Programming Reference Sheet

The DATA Step: Combining Data

- Combine tables by concatenating them (stacking), or matching them based on a variable
- Concatenating:
  - SAS reads all the rows from the first table listed on the set statement and writes them to the output table. Then from the second table, and so on
  - Columns with the same name are aligned
  - Columns not in all tables are included
  - The RENAME = option can rename columns in input tables, so they align in the output table
  - Additional DATA step statements can be used after the set statement to manipulate data

```
DATA output-dataset;
     set input-dataset1 input-dataset2
                (rename=(currentName = newName));
Run;
```

- Merging tables
  - All tables in the MERGE statement must be sorted by the column(s) listed in the BY statement
  - The MERGE statement combines rows where the BY-Column values match
  - Identify matching and no matching rows by using the IN= dataset option. IN variable values are 0 or 1.
    - 0 → table did NOT include the by-column value.
    - 1 → table did include the by-column value
  - Use a subsetting IF or IF-THEN logic to handle matching & nonmatching rows

```
DATA output-dataset;
     MERGE input-dataset1 <(in = VAR1)>
           input-dataset2 <(in = VAR2)>;
     BY by-column(s);
       < IF var1 = 1 and var2 = 1;>  /* all matching rows */
RUN;
```

The DATA Step: Functions

- SAS has functions to handle character, numeric and date columns.

```
new-var = function(argument1, argument2,…);
```

- Convert numeric values to character using the PUT function

```
char-var = put(numeric-var, format);
```

- Convert character values to numeric using the INPUT function

```
Numeric-var = input(char-var, informat);
```

- SAS has functions to handle character, numeric and date columns.
- Common Numeric Functions:

Function	What is does
RAND('distribution, paramter1,...)	Generates random numbers from a selected distribution
ROUND(number, <rounding unit>)	Rounds number to the nearest rounding unit (.01, .001, etc)
LARGEST(k, value1, value2, ...)	Returns the K^{th} largest non missing value
SUM(argument1, argument2,...)	Sums all non missing arguments

- Common Date Functions:

  NOTE: SAS Dates are numeric values calculated as the number of days since JAN 1, 1960.

Function	What is does
MDY(month, day, year)	Creates a SAS Date Value
TODAY()	Returns the current date as a numeric SAS date value
YEAR(date-var); MONTH(date-var) DAY(date-var) QTR(date-var)	Returns Year/Month/Day/QTR of the sas date value input
INTCK(interval, start-from, increment)	Increments a date/time/datetime value by a given time interval

- Common Character Functions

Function	What is does
TRIM(string)	Removes trailing blanks
STRIP(string)	Removes all leading and trailing blanks
SCAN(string, count, <char-list, <modifier>>)	Returns the nth word from a string
PROPCASE(string) UPCASE(string) LOWCASE(string)	Changes the casing of the string. Commonly used in statements of equality
SUBSTR(string, start-from, length)	Extracts a substring from the argument

Customizing SAS Output: Labels and Formats

- Labels and Formats can be applied in the DATA step and assigned as permanent attributes. These statements can also be used in reporting procedures as temporary attributes. (e.g. they need to be specified in each procedure)
- Labels can be used to provide more descriptive column headers. A label can include any text up to 256 characters.
- Add labels to more than one column in a single label statement

```
LABEL col-name1 = "Label Text 1"
      col-name2 = "Label Text 2" ;
```

- Formats are used to change the way values are displayed in data and reports.
- Formats do not change the underlying data values.
- Add formats to more than one column in a single statement

```
FORMAT date-var mmddyy10. num-var dollar13.2;
```

- Create your own custom formats using the **PROC FORMAT** procedure
  - **VALUE** statement specifies the criteria for creating one custom format.
  - Multiple VALUE statements can be used within the PROC FORMAT step.

```
PROC FORMAT;
     VALUE format-name <$>
           value-or-range-1 = 'formatted-value'
           value-or-range-2 = 'formatted-value'
           . . . ;
RUN;
```

SAS Base Programming Reference Sheet

Filtering Data

- WHERE Statements filter rows and can be used in both the DATA step and PROC steps.

```
WHERE expression;
```

- If the expression is true, rows are read, if false, they are not.
- WHERE statements can only work with columns that exist on an input dataset, not ones that are calculated during manipulation.
- Character values are case sensitive and must be in quotes " "
  EX: WHERE Car_Make = "Honda" will select different rows than WHERE car_make = "HONDA"
- Numeric values are not in quotes and can only include digits, decimal points and/or negative signs
- Compound conditions can be created using AND/OR
- Logic can be reversed with the NOT keyword
- Use the SAS Date constant when filtering with dates: "ddMONyyyy"d

```
WHERE Operators:
        = or EQ,
        ^=, ~= or NE
        > or GT
        >= or GE
        < or LT
        <= or LE
IN Operator
        WHERE col-name in (value1, value2,…);
        WHERE col-name NOT in (value1, value2,…);
Special Operators
        WHERE col-name IS MISSING
        WHERE col-name IS NOT MISSING
        WHERE col-name IS NULL
        WHERE col-name BETWEEN value1 AND
              value2
        WHERE col-name LIKE "value%"
        WHERE col-name LIKE "value_"
```

- A subsetting IF statement can be used on any variable that exists in the PDV. (e.g. variables on the input data set and new variables created)
- The expression used in the IF statement is written with most of the same operators as a WHERE expression.

```
/*Implicit output */
IF expression;
IF expression THEN output;

/* Explicit output to specific table*/
IF expression THEN output libref.output-dataset-name;
```

MACRO variables

- A macro variable stores a value that can be submitted into a SAS program
- If a macro variable is referenced inside quotation marks, then double quotation marks must be used
- Assign a value to a macro variable using a %LET statement
- The ampersand "&" must be used when calling a macro variable. The & triggers the macro facility

```
%LET macro-variable = value;
WHERE numvar = &macrovar;
WHERE charvar = "&macrovar";
```

Exporting Data

- Export a SAS dataset to variable file types (XLSX, TXT,CSV, etc) using a **PROC EXPORT** step
  - PROC EXPORT must be used to export unstructured data type. (e.g. CSV files)
  - **DMBS =** the database management system which specifies the type of data to export. (e.g. CSV, DLM, JMP, TAB)

```
PROC EXPORT DATA=input-table
                OUTFILE="output-file"
                <DBMS = identifier REPLACE> ;
RUN;
```

- Alternatively use a LIBNAME statement to export data.
- A LIBNAME statement can only be used if the output data type has an accessible SAS Engine (e.g. XLSX, JSON, XML).
- Ensure a LIBNAME libref CLEAR statement is used at the end to close the connection to the excel workbook.

```
LIBNAME myXL XLSX "C:/documents/Shopping.XLSX";
DATA myXL.shopping;
     SET work.shopping;
RUN;
LIBNAME myXL CLEAR
```

Exporting Reports

- The SAS Output Delivery System (ODS) can send reports to various file types to display reports including CSV, PowerPoint, RTF, and PDF.
- Each output type holds the same basic structure to open and close a file. Additional statements are available and optionable based on the file type.

```
ODS <destination> < destination specifications>;
        /* SAS Code that produces output */
ODS destination CLOSE;
```

- Additional options to excel files include:
  - Adding a style
  - Adding a worksheet label

```
ODS EXCEL FILE="filename.xlsx"
        STYLE=style
        OPTIONS(SHEET_NAME='label') ;
        /* SAS code that produces output on first
           worksheet */
        ODS EXCEL OPTIONS(SHEET_NAME='label');
        /* SAS code that  produces output on second
        worksheet */
ODS EXCEL CLOSE;
```

- PDF outputs can include a Table of Contents (PDFTOC) and Procedure labels in the bookmarks.

```
ODS PDF FILE="filename.xlsx"
        STYLE=style
        STARTPAGE = NO PDFTOC= 1;
        ODS PROCLABEL "label";
        /* SAS code that produces output */
ODS PDF CLOSE;
```

Additional Information

- For more information on SAS programming techniques, visit go.documentation.sas.com