**Summary of Lesson 1: Getting Started with SAS Programming**

**Exploring SAS**

[SAS](javascript:%20return%20false;) is a suite of business solutions and technologies to help organizations solve business problems. Base SAS is the centerpiece of all SAS software.   
  
It can be useful to look at SAS capabilities in a [simple framework](javascript:%20return%20false;):

* **Access data**: Using SAS, you can read any kind of data.
* **Manage data**: SAS gives you excellent data management capabilities
* **Analyze data**: For statistical analysis, SAS is the gold standard.
* **Present data**: You can use SAS to present your data meaningfully.

**Understanding the SAS Programming Process**

Here is the overall [process of programming in SAS](javascript:%20return%20false;).

1. **Define the business need**.
2. **Write a SAS program** based on the desired output, the necessary input, and the required processing.
3. **Run the program**.
4. **Review your results**.
5. If you find inaccuracies or errors, you **debug or modify** the program.

Depending on your results, you might need to repeat some of the steps.  
  
The power of SAS is that you can use it to read any type of data, including the following three major [file types](javascript:%20return%20false;):

* **Raw data files** contain data that has not been processed by any other computer program. They are text files that contain one record per line, and the record typically contains multiple fields. Raw data files aren’t reports; they are unformatted text.
* **SAS data sets** are specific to SAS. A SAS data set is data in a form that SAS can understand. Like raw data files, SAS data sets contain data. But in SAS data sets, the data is created only by SAS and can be read only by SAS.
* **SAS program files** contain SAS programming code. These instructions tell SAS how to process your data and what output to create. You can save and reuse SAS program files.

# Summary of Lesson 2: Working with SAS Programs

### Exploring SAS Programs

A [SAS program](javascript:%20return%20false;) consists of DATA steps and PROC steps. A [SAS programming step](javascript:%20return%20false;) is comprised of a sequence of statements. Every step has a beginning and ending step boundary. SAS compiles and executes each step independently, based on the step boundaries.

A SAS program can also contain global statements, which are outside DATA and PROC steps, and typically affect the SAS session. A TITLE statement is a global statement. After it is defined, a title is displayed on every report, unless the title is cleared or canceled.

[SAS statements](javascript:%20return%20false;) usually begin with an identifying keyword, and always end with a semicolon. SAS statements are free format and can begin and end in any column. A single statement can span multiple lines, and there can be more than one statement per line. Unquoted values can be lowercase, uppercase, or mixed case. This flexibility can result in programs that are difficult to read.

[Conventional formatting](javascript:%20return%20false;), also called structured formatting, uses consistent spacing to make a SAS program easy to read. To follow best practices, begin each statement on a new line, indent statements within each step, and indent subsequent lines in a multi-line statement.

[Comments](javascript:%20return%20false;) are used to document a program and to mark SAS code as non-executing text. There are [two types of comments](javascript:%20return%20false;): block comments and comment statements.

|  |
| --- |
| **/\***comment**\*/**  **\***comment statement**;** |

### Diagnosing and Correcting Syntax Errors

[Syntax errors](javascript:%20return%20false;) occur when program statements do not conform to the rules of the SAS language. Common syntax errors include misspelled keywords, missing semicolons, and invalid options. SAS finds syntax errors during the compilation phase, before it executes the program. When SAS encounters a syntax error, it writes the following to the log: the word ERROR or WARNING,the location of the error, and an explanation of the error. You should always check the log, even if the program produces output.

Mismatched or [unbalanced quotation marks](javascript:%20return%20false;) are considered a syntax error. In some programming environments, this results in a simple error message. In other environments, it is more difficult to identify this type of error.

## Sample Programs

**Submitting a SAS Program**

data work.newsalesemps;

set orion.sales;

where Country='AU';

run;

title 'New Sales Employees';

proc print data=work.newsalesemps;

run;

proc means data=work.newsalesemps;

class Job\_Title;

var Salary;

run;

title;

**Adding Comments to Your SAS Programs**

\*This program uses the data set orion.sales to create work.newsalesemps.;

data work.newsalesemps;

set orion.sales;

where Country='US';

run;

/\*

proc print data=work.newsalesemps;

run;\*/

proc means data=work.newsalesemps;

class Gender;

var Salary/\*numeric variable\*/;

run;

**Viewing and Correcting Syntax Errors**

daat work.newsalesemps;

length First\_Name $ 12

Last\_Name $ 18 Job\_Title $ 25;

infile "&path/newemps.csv" dlm=',';

input First\_Name $ Last\_Name $

Job\_Title $ Salary;

run;

proc print data=work.newsalesemps

run;

proc means data=work.newsalesemps average max;

class Job\_Title;

var Salary;

run;

# Summary of Lesson 3: Accessing Data

### Accessing SAS Libraries

SAS data sets are stored in SAS libraries. A [SAS library](javascript:%20return%20false;) is a collection of one or more SAS files that are recognized by SAS. SAS automatically provides one temporary and at least one permanent SAS library in every SAS session.

**Work** is a temporary library that is used to store and access SAS data sets for the duration of the session. **Sasuser** and **sashelp** are permanent libraries that are available in every SAS session.

You refer to a SAS library by a library reference name, or [libref](javascript:%20return%20false;). A libref is a shortcut to the physical location of the SAS files.

All SAS data sets have a [two-level name](javascript:%20return%20false;) that consists of the libref and the data set name, separated by a period. Data sets in the **work** library can be referenced with a one-level name, consisting of only the data set name, because **work** is the default library. Data sets in permanent libraries must be referenced with a two-level name.

You can create and access your own [SAS libraries](javascript:%20return%20false;). User-defined libraries are permanent but are not automatically available in a SAS session. You must assign a libref to a user-created library to make it available. You use a [LIBNAME statement](javascript:%20return%20false;) to associate the libref with the physical location of the library, that is, the physical location of your data. You can submit the LIBNAME statement alone at the start of a SAS session, or you can store it in a SAS program so that the SAS library is defined each time the program runs. If your program needs to reference data sets in multiple locations, you can use multiple LIBNAME statements.

|  |
| --- |
| **LIBNAME**libref***'***SAS***-***library***'*** <options>**;** |

Use [PROC CONTENTS](javascript:%20return%20false;) with libref**.**\_ALL\_ to display the contents of a SAS library. The report will list all the SAS files contained in the library, as well as the descriptor portion of each data set in the library. Use the NODS option in the PROC CONTENTS statement to suppress the descriptor information for each data set.

|  |
| --- |
| **PROC CONTENTS DATA**=libref**.\_ALL\_ NODS; RUN;** |

After associating a libref with a permanent library, you can write a [PROC PRINT](javascript:%20return%20false;) step to display a SAS data set within the library.

|  |
| --- |
| **PROC PRINT DATA=**libref***.***SAS-data-set**;** **RUN;** |

In an interactive SAS session, a libref remains in effect until you cancel it, change it, or end your SAS session. To [cancel a libref](javascript:%20return%20false;), you submit a LIBNAME statement with the CLEAR option. This clears or disassociates a libref that was previously assigned. To specify a different physical location, you submit a LIBNAME statement with the same libref name but with a different filepath.

|  |
| --- |
| **LIBNAME**libref**CLEAR;** |

When a SAS session ends, everything in the **work** library is deleted. The librefs are also deleted. Remember that the contents of permanent libraries still exist in in the operating environment, but each time you start a new SAS session, you must resubmit the LIBNAME statement to redefine a libref for each user-created library that you want to access.

### Examining SAS Data Sets

[SAS data sets](javascript:%20return%20false;) are specially structured data files that SAS creates and that only SAS can read. A SAS data set is displayed as a table composed of variables and observations. A SAS data set contains a descriptor portion and a data portion.

The descriptor portion contains general information about the data set (such as the data set name and the number of observations) and information about the variable attributes (such as name, type, and length). There are two types of variables: character and numeric. A character variable can store any value and can be up to 32,767 characters long. Numeric variables store numeric values in floating point or binary representation in 8 bytes of storage by default. Other attributes include formats, informats, and labels. You can use PROC CONTENTS to browse the descriptor portion of a data set.

|  |
| --- |
| **PROC CONTENTS DATA**=libref**.**SAS-data-set**; RUN;** |

The data portion contains the data values. Data values are either character or numeric. A valid value must exist for every variable in every observation in a SAS data set. A [missing value](javascript:%20return%20false;) is a valid value in SAS. A missing character value is displayed as a blank, and a missing numeric value is displayed as a period. You can specify an alternate character to print for missing numeric values using the MISSING= SAS system option. You can use PROC PRINT to display the data portion of a SAS data set.

[SAS variable and data set names](javascript:%20return%20false;) must be 1 to 32 characters in length and start with a letter or underscore, followed by letters, underscores, and numbers. Variable names are not case sensitive.

## Sample Programs

**Accessing a SAS Library**

/\*Replace filepath with the physical location of your practice files.\*/

%let path=filepath;

libname orion "&path";

**Browsing a Library**

proc contents data=orion.\_all\_;

run;

proc contents data=orion.\_all\_ nods;

run;

**Viewing a Data Set with PROC PRINT**

proc print data=orion.country;

run;

**Viewing the Descriptor Portion of a Data Set**

proc contents data=orion.sales;

run;

**Viewing the Data Portion of a SAS Data Set**

proc print data=orion.sales;

run;

# Summary of Lesson 4: Producing Detail Reports

### Subsetting Report Data

You can use the [VAR statement](javascript:%20return%20false;)in a PROC PRINT step to subset the variables in a report. You specify the variables to include and list them in the order in which they are to be displayed.

You can use the [SUM statement](javascript:%20return%20false;) in a PROC PRINT step to calculate and display report totals for the requested numeric variables.

|  |
| --- |
| **PROC PRINT DATA**=SAS-data-set**;**          **VAR** variable(s)**;**  **SUM** variable(s)**;**  **RUN;** |

The [WHERE statement](javascript:%20return%20false;) in a PROC PRINT step subsets the observations in a report. When you use a WHERE statement, the output contains only the observations that meet the conditions specified in the WHERE expression. This expression is a sequence of operands and operators that form a set of instructions that define the condition. The operands can be constants or variables. Remember that variable operands must be defined in the input data set. Operators include [comparison, arithmetic](javascript:%20return%20false;), [logical](javascript:%20return%20false;), and [special WHERE operators](javascript:%20return%20false;).

|  |
| --- |
| **WHERE**where-expression**;** |

You can use the [ID statement](javascript:%20return%20false;) in a PROC PRINT step to specify a variable to print at the beginning of the row instead of an observation number. The variable that you specify replaces the Obs column.

|  |
| --- |
| **ID**variable(s)**;** |

### Sorting and Grouping Report Data

The [SORT procedure](javascript:%20return%20false;) sorts the observations in a data set. You can sort on one variable or multiple variables, sort on character or numeric variables, and sort in ascending or descending order. By default, SAS replaces the original SAS data set unless you use the OUT= option to specify an output data set. PROC SORT does not generate printed output.

Every PROC SORT step must include a BY statement to specify one or more BY variables. These are variables in the input data set whose values are used to sort the data. By default, SAS sorts in ascending order, but you can use the keyword DESCENDING to specify that the values of a variable are to be sorted in descending order. When your SORT step has [multiple BY variables](javascript:%20return%20false;), some variables can be in ascending and others in descending order.

You can also use a [BY statement](javascript:%20return%20false;) in PROC PRINT to display observations grouped by a particular variable or variables. The groups are referred to as BY groups. Remember that the input data set must be sorted on the variables specified in the BY statement.

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

### Enhancing Reports

You can enhance a report by adding titles, footnotes, and column labels. Use the global [TITLE statement](javascript:%20return%20false;) to define up to 10 lines of titles to be displayed at the top of the output from each procedure. Use the global FOOTNOTE statement to define up to 10 lines of footnotes to be displayed at the bottom of the output from each procedure.

|  |
| --- |
| **TITLE**n'text'**; FOOTNOTE**n 'text'**;** |

Titles and footnotes remain in effect until you [change or cancel](javascript:%20return%20false;) them, or until you end your SAS session. Use a null TITLE statement to cancel all titles, and a null FOOTNOTE statement to cancel all footnotes.

Use the [LABEL statement](javascript:%20return%20false;) in a PROC PRINT step to define temporary labels to display in the report instead of variable names. Labels can be up to 256 characters in length. Most procedures use labels automatically, but PROC PRINT does not. Use the LABEL option in the PROC PRINT statement to tell SAS to display the labels. Alternatively, the [SPLIT=](javascript:%20return%20false;) option tells PROC PRINT to use the labels and also specifies a split character to control line breaks in column headings.

|  |
| --- |
| **PROC PRINT DATA**=SAS-data-set **LABEL;**          **LABEL**variable='label'                      variable='label'                    ...**; RUN;** |

|  |
| --- |
| **SPLIT**='split-character'**;** |

## Sample Programs

**Subsetting Your Report**

proc print data=orion.sales;

var Last\_Name First\_Name Salary;

sum Salary;

run;

**Selecting Observations**

proc print data=orion.sales noobs;

var Last\_Name First\_Name Salary Country;

where Country='AU' and Salary<25500;

run;

**Using the CONTAINS Operator**

proc print data=orion.sales noobs;

var Last\_Name First\_Name Country Job\_Title;

where Country='AU' and Job\_Title contains 'Rep';

run;

**Subsetting Observations and Replacing the Obs Column**

proc print data=orion.customer\_dim;

where Customer\_Age=21;

id Customer\_ID;

var Customer\_Name

Customer\_Gender Customer\_Country

Customer\_Group Customer\_Age\_Group

Customer\_Type;

run;

**Sorting a Data Set**

proc sort data=orion.sales

out=work.sales\_sort;

by Salary;

run;

proc print data=work.sales\_sort;

run;

**Sorting a Data Set by Multiple Variables**

proc sort data=orion.sales

out=work.sales2;

by Country descending Salary;

run;

proc print data=work.sales2;

run;

**Grouping Observations in Report**s

proc sort data=orion.sales

out=work.sales2;

by Country descending Salary;

run;

proc print data=work.sales2;

by Country;

run;

**Displaying Titles and Footnotes in a Report**

title1 'Orion Star Sales Staff';

title2 'Salary Report';

footnote1 'Confidential';

proc print data=orion.sales;

var Employee\_ID Last\_Name Salary;

run;

proc print data=orion.sales;

var Employee\_ID First\_Name Last\_Name Job\_Title Hire\_Date;

run;

**Changing and Canceling Titles and Footnotes**

title1 'Orion Star Sales Staff';

title2 'Salary Report';

footnote1 'Confidential';

proc print data=orion.sales;

var Employee\_ID Last\_Name Salary;

run;

title1 'Employee Information';

proc print data=orion.sales;

var Employee\_ID First\_Name Last\_Name Job\_Title Hire\_Date;

run;

**Displaying Labels in a Report**

title1 'Orion Star Sales Staff';

title2 'Salary Report';

footnote1 'Confidential';

proc print data=orion.sales label;

var Employee\_ID Last\_Name Salary;

label Employee\_ID = 'Sales ID'

Last\_Name = 'Last Name'

Salary = 'Annual Salary';

run;

title;

footnote;

# Summary of Lesson 5: Formatting Data Values

### Using SAS Formats

A [format](javascript:%20return%20false;) is an instruction that tells SAS how to display data values in output reports. You can add a FORMAT statement to a PROC PRINT step to specify temporary SAS formats that control how values appear in the report. There are many existing SAS formats that you can use. Character formats begin with a dollar sign, but numeric formats do not.

|  |
| --- |
| **FORMAT**variable(s) format**;** |

SAS stores date values as the number of days between January 1, 1960, and a specific date. To make the dates in your report recognizable and meaningful, you must apply a [SAS date format](javascript:%20return%20false;) to the SAS date values.

### Creating and Applying User-Defined Formats

You can create your own [user-defined formats](javascript:%20return%20false;). When you create a user-defined format, you don't associate it with a particular variable or data set. Instead, you create it based on values that you want to display differently. The formats will be available for the remainder of your SAS session. You can apply user-defined formats to a specific variable in a PROC PRINT step.

You use the [FORMAT procedure](javascript:%20return%20false;) to create a format. You assign a format name that can have up to 32 characters. The name of a character format must begin with a dollar sign, followed by a letter or underscore, followed by letters, numbers, and underscores. Names for numeric formats must begin with a letter or underscore, followed by letters, numbers, and underscores. A format name cannot end in a number and cannot be the name of a SAS format.

You use a [VALUE statement](javascript:%20return%20false;) in a PROC FORMAT step to specify the way that you want the data values to appear in your output. You define value-range sets to specify the values to be formatted and the formatted values to display instead of the stored value or values. The value portion of a value-range set can include an individual value, a range of values, a list of values, or a keyword. The keyword OTHER is used to define a value to display if the stored data value does not match any of the defined value-ranges.

|  |
| --- |
| **PROC FORMAT;**         **VALUE**format-name value-or-range1='formatted-value1'                                            value-or-range2='formatted-value2'                                           ...***;*** **RUN;** |

When you define a numeric format, it is often convenient to use numeric ranges in the value-range sets. Ranges are inclusive by default. To[exclude the endpoints](javascript:%20return%20false;), use a less-than symbol after the low end of the range or before the high end.

The [LOW and HIGH](javascript:%20return%20false;) keywords are used to define a continuous range when the lowest and highest values are not known. Remember that for character values, the LOW keyword treats missing values as the lowest possible values. However, for numeric values, LOW does not include missing values.

## Sample Programs

**Applying Temporary Formats**

proc print data=orion.sales label noobs;

where Country='AU' and

Job\_Title contains 'Rep';

label Job\_Title='Sales Title'

Hire\_Date='Date Hired';

var Last\_Name First\_Name Country Job\_Title

Salary Hire\_Date;

run;

proc print data=orion.sales label noobs;

where Country='AU' and

Job\_Title contains 'Rep';

label Job\_Title='Sales Title'

Hire\_Date='Date Hired';

format Hire\_Date mmddyy10. Salary dollar8.;

var Last\_Name First\_Name Country Job\_Title

Salary Hire\_Date;

run;

**Specifying a User-Defined Format for a Character Variable**

proc format;

value $ctryfmt 'AU'='Australia'

'US'='United States'

other='Miscoded';

run;

proc print data=orion.sales label;

var Employee\_ID Job\_Title Salary

Country Birth\_Date Hire\_Date;

label Employee\_ID='Sales ID'

Job\_Title='Job Title'

Salary='Annual Salary'

Birth\_Date='Date of Birth'

Hire\_Date='Date of Hire';

format Salary dollar10.

Birth\_Date Hire\_Date monyy7.

Country $ctryfmt.;

run;

**Specifying a User-Defined Format for a Numeric Variable**

proc format;

value tiers low-<50000='Tier1'

50000-100000='Tier2'

100000<-high='Tier3';

run;

proc print data=orion.sales;

var Employee\_ID Job\_Title Salary

Country Birth\_Date Hire\_Date;

format Birth\_Date Hire\_Date monyy7.

Salary tiers.;

run;

# Summary of Lesson 6: Reading SAS Data Sets

### Reading a SAS Data Set

You use a [DATA step](javascript:%20return%20false;) to create a new SAS data set from an existing SAS data set. The DATA step begins with a DATA statement, which provides the name of the SAS data set to create. Include a SET statement to name the existing SAS data set to be read in as input.

You use the WHERE statement to subset the input data set by selecting only the observations that meet a particular condition. To subset based on a SAS date value, you can use a [SAS date constant](javascript:%20return%20false;) in the WHERE expression. SAS automatically converts a date constant to a SAS date value.

|  |
| --- |
| **DATA**output-SAS-data-set**;        SET**input-SAS-data-set***;***        **WHERE** where-expression**;**  **RUN;** |

You use an [assignment statement](javascript:%20return%20false;) to create a new variable. The assignment statement evaluates an expression and assigns the resulting value to a new or existing variable. The expression is a sequence of operands and operators. If the expression includes arithmetic operators, SAS performs the numeric operations based on priority, as in math equations. You can use parentheses to clarify or alter the order of operations.

|  |
| --- |
| variable=expression**;** |

### Customizing a SAS Data Set

By default, the SET statement reads all of the observations and variables from the input data set and writes them to the output data set. You can customize the new data set by selecting only the observations and variables that you want to include. You can use a WHERE statement to select the observations, as long as the variables included in the condition come from the input data set. You can use a [DROP statement](javascript:%20return%20false;) to list the variables to exclude from the new data set, or use a KEEP statement to list the variables to include. If you use a KEEP statement, you must include every variable to be written, including any new variables.

|  |
| --- |
| **DROP**variable-list**; KEEP**variable-list**;** |

SAS processes the DATA step in two phases: the [compilation phase](javascript:%20return%20false;)and the [execution phase](javascript:%20return%20false;).

You can [subset](javascript:%20return%20false;) the original data set with a WHERE statement for variables that are defined in the input data set, and a [subsetting IF statement](javascript:%20return%20false;) for new variables that are created in the DATA step. Remember that, although IF expressions are similar to WHERE expressions, you cannot use special WHERE operators in IF expressions.

|  |
| --- |
| **IF**expression**;** |

To [subset observations](javascript:%20return%20false;) in a PROC step, you must use a WHERE statement. You cannot use a subsetting IF statement in a PROC step. To subset observations in a DATA step, you can always use a subsetting IF statement. However, a WHERE statement can make your DATA step more efficient because it subsets on input.

### Adding Permanent Attributes

When you use the [LABEL statement](javascript:%20return%20false;) in a DATA step, SAS permanently associates the labels to the variables by storing the labels in the descriptor portion of the data set. Using a [FORMAT statement](javascript:%20return%20false;) in a DATA step permanently associates formats with variables. The format information is also stored in the descriptor portion of the data set. You can use [PROC CONTENTS](javascript:%20return%20false;) to view the label and format information. PROC PRINT does not display permanent labels unless you use the LABEL or SPLIT= option.

|  |
| --- |
| **LABEL**variable='label'            variable='label'              ...**;** |

|  |
| --- |
| **FORMAT**variable(s) format ...**;** |

## Sample Programs

**Subsetting Observations in the DATA Step**

proc print data=orion.sales;

run;

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep';

run;

proc print data=work.subset1;

run;

**Subsetting Observations and Creating a New Variable**

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep' and

Hire\_Date<'01jan2000'd;

Bonus=Salary\*.10;

run;

proc print data=work.subset1 noobs;

var First\_name Last\_Name Salary

Job\_Title Bonus Hire\_Date;

format Hire\_Date date9.;

run;

**Subsetting Variables in a DATA Step: DROP and KEEP**

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep';

Bonus=Salary\*.10;

drop Employee\_ID Gender Country Birth\_Date;

run;

proc print data=work.subset1;

run;

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep';

Bonus=Salary\*.10;

keep First\_Name Last\_Name Salary Job\_Title Hire\_Date Bonus;

run;

proc print data=work.subset1;

run;

**Selecting Observations by Using the Subsetting IF Statement**

data work.auemps;

set orion.sales;

where Country='AU';

Bonus=Salary\*.10;

if Bonus>=3000;

run;

proc print data=work.auemps;

run;

**Adding Permanent Labels to a SAS Data Set**

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep';

Bonus=Salary\*.10;

label Job\_Title='Sales Title'

Hire\_Date='Date Hired';

drop Employee\_ID Gender Country Birth\_Date;

run;

proc contents data=work.subset1;

run;

proc print data=work.subset1 label;

run;

**Adding Permanent Formats to a SAS Data Set**

data work.subset1;

set orion.sales;

where Country='AU' and

Job\_Title contains 'Rep';

Bonus=Salary\*.10;

label Job\_Title='Sales Title'

Hire\_Date='Date Hired';

format Salary Bonus dollar12.

Hire\_Date ddmmyy10.;

drop Employee\_ID Gender Country Birth\_Date;

run;

proc contents data=work.subset1;

run;

proc print data=work.subset1 label;

run;