# LEARNING sas

#sas

# Table of Contents

# About

# Chapter 1: Getting started with sas

## Examples

### Installation or Setup

SAS can be run in client-server model, using either the Enterprise Guide thick client or the SAS Studio thin (web-enabled) client, or in "local server" mode where a fully functional SAS system is present on a local machine (Windows or Unix/Linux desktop or server running in interactive mode) and run either in Display Manager mode (the local client) or through one of the client-server clients above (connected to the locally installed server).

SAS installation typically is performed by a SAS administrator, who will install the software from a software depot that is customized for the site (and often provided by SAS Institute directly).

For the purpose of learning SAS, there is also the free SAS University Edition, which can be installed for free for educational purposes by anyone on a Windows, Mac, or Unix/Linux computer. It is available from SAS directly, currently at the SAS University Edition page, either by running an AWS instance (on the free tier) or by downloading a virtual machine locally. See the installation guide on SAS.com for up to date instructions, or below for the current (July 2016) instructions.

To install it locally, you first download and install Oracle Virtualbox 5.0 (Windows/Mac/Linux). Then download the newest SAS University Edition disk image, which is around 2GB and requires setting up a SAS.com profile.

Once you've done so, you need to set up the virtual machine in VirtualBox. Import the SAS VM as an appliance ("Import Appliance" in VirtualBox). Create a folder for SAS to use as its local storage (so you can put files in a location SAS can see them), and set that up as a Shared Folder in the machine settings dialog box. Set it up to auto-mount.

Then, start the SAS virtual machine, and once it's started you can connect via your web browser, connecting to http://localhost:10080/ if you used the default settings.

If you have issues, the SAS Community Forums - Analytics U are the vendor forums to obtain support, or ask a question on Stack Overflow.

### Overview of Base SAS

SAS is an integrated system of software solutions that enables you to perform the following tasks:

- data entry, retrieval, and management
- report writing and graphics design
- statistical and mathematical analysis
- business forecasting and decision support
- operations research and project management
- applications development

How you use SAS depends on what you want to accomplish. Some people use many of the capabilities of the SAS System, and others use only a few.

## Hello World!

Due to the stucture of SAS, there are three main ways to create "Hello World!" examples:

1. Within a data step to put a message into the SAS log (`_null_` denotes that no output dataset should be created):

```
data _null_;
    put "Hell" "o World!";
run;
```

2. Within a data step to store "Hello World!" within a variable (`foo` denotes that an output dataset called `foo` should be created that a) contains only one record and b) contains only one variable: `bar`, which has a value of `Hello World!`):

```
data foo ;
    bar="Hello" ;
    put bar= "World!";
run ;
```

3. Via the SAS Macro language (in 'open code' outside of any data steps). `&` identifies a call to a macro variable and `.` identifies the end of the variable (if a white space character is not wanted):

```
%let foo=Hello;
%put &foo.o World!;
```

4. Hybrid: Using a macro variable within a data step:

```
%let foo=Hello;

data _null_ ;
  put "&foo World!";
run ;
```

## SAS Server Architecture

**Overview**: There are typically two types of SAS Deployments:

1. SAS Foundation only installation (BASE SAS). This is typically is installed on a PC. It does not run any server software.

2. SAS Planned Deployment for their server architecture which will install the SAS server environment along with possibly any SAS client software.

Which one of these you have will be indicated in your SAS Software Order email by indicating planning or non-planning. If you are doing a planned installation you will need a plan file for your order that first your topology.

Installation Note 44320: Using deployment plans during a SAS® installation

**SAS Server Architecture**

The SAS server environment is broken into 3 different tiers:

1. SAS Metadata Server(s) - The SAS Metadata server is responsible for managing the SAS server environment including libraries, users, and server configuration.

**2. SAS Application Server(s)** - The SAS Application Server is mostly a compute server where your clients would typically launch jobs from.

**3. SAS Middle Tier (s)** = The SAS Middle Tier is primarily your Web tier which runs your web applications.

**4. Client Tier** - The client tier is your users client applications they use to connect to the environment such as SAS Enterprise Guide.

Paper 363-2011| Understanding the Anatomy of a SAS® Deployment: What's in My Server Soup? Mark Schneider, Donna Bennett, and Connie Robison, SAS Institute Inc., Cary, NC

**Topology:**

The SAS Metadata Tier, SAS Application Server tier, and SAS Middle Tier can be installed on a single machine server, or spread out across multiple servers. This is determined by the plan file you have, it should meet the desired topology for your deployment.

Typically most, if not all of the client tier are Windows based applications, so the client tier would be on your SAS users workstations. Optionally they could probably be installed on the server(s) as well if they are Windows based.

SAS Supported Operating Systems

## Versioning

The main current versions of SAS are 9.4 and 9.3 these are the versions of the base SAS engine most commonly used today. The link to release notes for versions 9.1 + and other related documentation are included below.

***Please note also, there are various packages and functions which extend the functionality of SAS, and these have their own self standing documentation and functionality*.**

- SAS 9.4 - Key Documentation and Release Notes
- SAS 9.3 - Key Documentation and Release Notes
- SAS 9.2 - Key Documentation and Release Notes
- SAS 9.1.x - Key Documentation and Release Notes

Read Getting started with sas online: https://riptutorial.com/sas/topic/2108/getting-started-with-sas

# Chapter 2: Copy a file, byte for byte

## Introduction

If you're using SAS to produce reporting of some sort, you're going to find yourself needing to copy a file at some point. I've mostly used this method for copying an excel template, and then dumping data via PROC EXPORT into the new file I've created.

This is a great example I've found from Chris Hemedinger ( http://blogs.sas.com/content/sasdummy/2011/06/17/how-to-use-sas-data-step-to-copy-a-file-from-anywhere/).

## Examples

### Copying any file, byte by byte

```
/* these IN and OUT filerefs can point to anything */
filename in "anyfilehere.xlsx";
filename out "anyfilehere.xlsx";


/* copy the file byte-for-byte  */
data _null_;
  length filein 8 fileid 8;
  filein = fopen('in','I',1,'B');
  fileid = fopen('out','O',1,'B');
  rec = '20'x;
  do while(fread(filein)=0);
     rc = fget(filein,rec,1);
     rc = fput(fileid, rec);

     rc =fwrite(fileid);
  end;
  rc = fclose(filein);
  rc = fclose(fileid);
run;

filename in clear;
filename out clear;
```

Read Copy a file, byte for byte online: https://riptutorial.com/sas/topic/9394/copy-a-file--byte-for-byte

# Chapter 3: Creating Macro Variables

## Introduction

Using Macro Variables throughout your SAS programs is a basic functionality that every SAS programmer must be familiar with. Using Macro Variables can help you to keep your code simple and generic. Generic code is reusable code.

## Examples

### Using %LET

I would describe %LET as being the most simple way to creating a Macro Variable in SAS.

```
%LET variableName = variableValue;
```

Now, anywhere you use `&variableName`, it will resolve to `variableValue`.

> NOTE:you may want to consider that `variableValue` all on its own might bring you syntax errors, depening on what the value is and how it's used. For example, if it is a date and you're using it in the WHERE of a PROC SQL statement, it will need to be written as `"&variableName"d` to work properly.

### Using PROC SQL

Using PROC SQL is a good way to get quick results from a table and throw them into variables. I usually find that when I want to get a count of records I just loaded to a table, I can get that count into a variable with a quick PROC SQL call.

```
PROC SQL;
SELECT
    COUNT(*) INTO:aVariable
FROM
    MyTable

;QUIT;
```

In the example above, `aVariable` will represent how many records exist in `MyTable`.

You can also use PROC SQL for creating multiple Macro Variables.

```
PROC SQL;
SELECT
    a,
    b,
    c INTO:aVariable, :bVariable, :cVariable
FROM
    MyTable
```

```
;QUIT;
```

In the example above, the variables created in the INTO statement will match up to the columns pulled in the order they are returned from the SELECT statement. However, only the first row of results will be used to fill those 3 variables.

If you want to store more than a single row, and you're on version 6.11 or beyond, use the following example:

```
PROC SQL;
    SELECT DISTINCT
        a,
        b,
        c INTO :aVariable1 - :aVariable5,
               :bVariable1 - :bVariable5,
               :cVariable1 - :cVariable5
    FROM
        MyTable
;QUIT;
```

> The keywords `THROUGH` and `THRU` can he used en lieu of the dash `-`

## Using Call Symput() in a DATA step

```
DATA _null_;
            CALL SYMPUT('testVariable','testValueText');
;RUN;
```

In the example above, `%PUT &testVariable;` will resolve to `testvalueText`.

You may find the need to format your variable within the SYMPUT() call.

```
DATA _null_;
            CALL SYMPUT('testDate',COMPRESS(PUT(today(),date9.)));
;RUN;
```

In the example above,`%PUT &testDate;` will resolve to `10MAR2017`

Read Creating Macro Variables online: https://riptutorial.com/sas/topic/9403/creating-macro-variables

# Chapter 4: data step

## Examples

### getting data with data setp

```
data newclass(keep=first_name sex weight yearborn);
  set sashelp.class(drop=height rename=(name=first_name));
  yearborn=year(date())-age;
  if yearborn >2002;
run;
```

Data specifies the target data set. Keep option specifies columns to print to target.

Set specifies source data set. Drop specifies columns not to take. Rename renames name to first_name.

Yearborn is a calculated implicit numeric variable (column).

Filter and implicit output data with `if` for pupils born after 2002.

Read data step online: https://riptutorial.com/sas/topic/10673/data-step

# Chapter 5: DO Loop

## Examples

### DO Loop

```
DATA salary;
    /*define variables*/
    raise=0.1;
    salary=50000;
    year=1;
    /*do loop*/
    DO year=1 to 20 by 2;
        salary + salary*raise;
        output; /*generates an observation for each iteration of the do loop, optional*/
    END;
RUN;
```

### Macro do loop

```
%macro doloop;
  %do age=11 %to 15 %by 2;
    title Age=&age.;
    proc print data=sashelp.class(where=(age=&age.));
    run;
  %end;
%mend;
%doloop;
```

Read DO Loop online: https://riptutorial.com/sas/topic/7919/do-loop

# Chapter 6: Informats in SAS

## Introduction

SAS `informats` instruct SAS on how to read data from any input location (such as a file, an excel spreadsheet, a named pipe, or even another SAS variable, etc.) into a variable.

SAS has just two data types - character and numeric, and each informat is specific to storing the value into either a character or numeric variable. If the destination variable is a character, then the informat will begin with a `$` symbol, anything else will be a numeric informat.

## Remarks

Informats are very important especially when we import data from other datasets. For example, most of the times while working on real time data, we extract data from various data sources (Oracle,Mysql,Teradata etc). Every time we import data we need to specify the informat statement so SAS can read the data properly.

## Examples

### Importing excel data into SAS

For example, say below is the sample data in an Excel 'Test',

```
Purchase_Date    Customer_Name    Price
05-05-2017    Adam    1075
06-05-2017    Noah    1093
07-05-2017    Peter    1072
08-05-2017    Louis    1101
09-05-2017    Zoe    1248
10-05-2017    Kevin    1045
11-05-2017    Messiah    1072
12-05-2017    John    1046
13-05-2017    Stephen    1043
14-05-2017    Solly    1113
15-05-2017    Jeevan    1137
```

You should use the below code to import this successfully,

```
Data Test;
Infile 'D:\Test.csv';
Delimiter=',' Missover DSD Getnames=Yes;
Informat Purchase_Date date9.;
Informat Price dollarx10.2;
Format Purchase_Date date9.;
Format Price dollarx10.2;
run;
```

```
Informat in the above code helps SAS to read the data from Excel.
Format in the above code helps to write the data properly into SAS Data set.
```

## Importing character vs numeric

The example below uses the input statement to read a value from a source (in this case the string `123`) into a both a character destination and a numeric destination.

```
data test;
   source = '123';
   numeric_destination = input(source, best.);
   character_destination = input(source, $3.);
run;
```

Read Informats in SAS online: https://riptutorial.com/sas/topic/9888/informats-in-sas

# Chapter 7: Proc SQL

## Examples

### Create an empty dataset based on an existing dataset

Method 1:

```
 proc sql;
    create table foo like sashelp.class;
    quit;
```

Method 2:

```
proc sql;
create table bar as
    select * from sashelp.class (obs=0);
quit;
```

Method 1 should be the preferred option

### SELECT Syntax

```
PROC SQL options;
  SELECT column(s)
FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
ORDER BYcolumn(s);
QUIT;
```

Example 1:

```
proc sql;
select name
       ,sex
    from sashelp.class ;
quit;
```

The SELECT statement is specified in this order :

```
1.select;
2.from;
3.where;
4.group by;
5.having;
6.order by.
```

"select" and "from" are required. The other clauses are optional.

Read Proc SQL online: https://riptutorial.com/sas/topic/5870/proc-sql

# Chapter 8: Reading Data

## Introduction

Reading data into a SAS dataset can be accomplished using multiple approaches including the `datalines` statement, from an external file using an `infile` statement in the data step, or reading data from an external file using `proc import`. In addition you can read in data from external sources that are odbc compliant (e.g. SQL databases) using the odbc drivers.

## Examples

### Read text file with comma delimiter

```
DATA table-name;
    INFILE "file-path/file-name.csv" dsd;
    INPUT Name $ City $ Age;
RUN;
```

### Read data from excel file

```
PROC IMPORT DATAFILE = "file-path/file-name.xlsx" OUT=data_set DBMS=XLSX REPLACE;
```

### PROC IMPORT for Excel, importing a specific sheet

There will be times where you only want to import a specific sheet from an excel file with multiple sheets. To do that, we'll use "**SHEET=**".

```
PROC IMPORT
    OUT= YourNewTable
    DATAFILE= "myfolder/excelfilename.xlsx"
    DBMS=xlsx
    REPLACE;
    SHEET="Sheet1";
    GETNAMES=YES;
RUN;
```

> Also take note of the ability to specify whether or not the top row imported contains column names or not (**GETNAMES=YES** (or NO).

Read Reading Data online: https://riptutorial.com/sas/topic/7989/reading-data

# Chapter 9: Resolving Macro Variables in quotes within PROC SQL Pass-throughs

## Introduction

One of the challenges I faced when I first started using SAS was not only passing Macro Variable data into a PROC SQL pass-through, but having it resolve properly if it needed quotes around it. When passing a string like value or date/datetime into a PROC SQL pass-through, it most likely needs to have single quotes around it when it resolves.

I have found the best results when using the %BQUOTE function to accomplish this.

## Remarks

More information on the %BQUOTE function can be found here: https://v8doc.sas.com/sashtml/macro/z4bquote.htm

## Examples

### Pass-through with Macro Variable that is a Date

First, I will place my date into a Macro Variable.

> NOTE: I find that date9. works great with IBM® Netezza® SQL and Transact-SQL. Use whichever format that works for the type of SQL you're executing.

```
data _null_;
            call symput('testDate',COMPRESS(put(today(),date9.)));
;RUN;
%PUT &testDate;
```

My %PUT statement resolves to: 10MAR2017

Next, I want to run a PROC SQL Pass-through and resolve that Macro Variable inside to specify a date.

```
PROC SQL;
CONNECT TO odbc AS alias (dsn=myServer user=userName password= pass);
CREATE TABLE TableName AS
SELECT *
FROM connection to alias
    (
        SELECT *
        FROM
            Database.schema.MyTable
        WHERE
            DateColumn = %bquote('&testDate')
    );
QUIT;
```

%bquote('&testDate') will resolve to '10MAR2017' when the code executes.

Read Resolving Macro Variables in quotes within PROC SQL Pass-throughs online: https://riptutorial.com/sas/topic/9396/resolving-macro-variables-in-quotes-within-proc-sql-pass-throughs

# Chapter 10: SAS Formats

## Introduction

Informats and formats are used to tell SAS how to read and write the data respectively. Informats are commonly used in a datastep when reading data from an external file. Informats are rarely used in PROCs. Formats are commonly used in both data steps and PROCs.

## Remarks

SAS Formats convert either numeric or character values to character values. A format can either be applied using a `format` or `put` statement, which changes the way a value is displayed, or using the `put` function to store the formatted value in a new variable.

---

There are four categories of formats :

- Character - instructs SAS to write character data values from character variables.
- Date and Time - instructs SAS to write data values from variables that represent dates, times, and datetimes.
- ISO 8601 - instructs SAS to write date, time, and datetime values using the ISO 8601 standard.
- Numeric - instructs SAS to write numeric data values from numeric variables.

---

Formats usually take the form `<formatname><w>.<d>;`, `w` being the width (including any decimals and the point), `d` being the number of decimal places.

---

Common date formats (applied to SAS date values) :

- `date9.` e.g. 02AUG2016
- `ddmmyyn8.` e.g. 02082016
- `ddmmyy8.` e.g. 02/08/16
- `yymmdd10.` e.g. 20160802
- `year4.` e.g. 2016

Common numeric formats (applied to numbers) :

- `comma11.0` e.g. 1,234,567
- `comma12.2` e.g. 1,234,567.00
- `dollar11.2` e.g. $5,789.12
- `nlmnlgbp11.2` e.g. £2,468.02

Other formats :

- `$hex8.`, convert string to hex

- `$upcase.`, convert string to upper-case
- `$quote.`, enclose a string in quotes

A full list of formats can be found here > https://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a001263753.htm

## Examples

### Using the format statement

The `format` statement applies the given format to the specified variable *for display purposes only*, i.e. the underlying value does not change.

```
data example1 ;
  Date  = '02AUG2016'd ; /* stored as a SAS date, i.e. a number */
  Date2 = '31AUG2016'd ;
  format Date monyy7. Date2 yymmddn8. ;
run ;
```

| Date | Date2 |
|---|---|
| AUG2016 | 20160831 |

### Using the format statement to group data

You can apply formats within a procedure, e.g. to change the groupings within a `proc summary` or `proc freq`.

---

Grouping SAS dates

```
data example2 ;
  do Date = '01JUN2016'dt to '31AUG2016'dt ;
    Days = 1 ;
    output ;
  end ;
run ;

/* Summarise by year & month */
proc summary data=example2 nway ;
  class Date ;
  var Days ;
  output out=example2_sum (drop=_TYPE_ _FREQ_) sum= ;
  format Date yymmn6. ; /* e.g. 201606 */
run ;
```

| Date | Days |
|---|---|
| 201606 | 30 |
| 201607 | 31 |

| Date | Days |
|---|---|
| 201608 | 31 |

```
/* Summarise by month & year */
proc summary data=example2 nway ;
  class Date ;
  var Days ;
  output out=example2_sum2 (drop=_TYPE_ _FREQ_) sum= ;
  format Date monyy7. ; /* e.g. JUN2016 */
run ;
```

| Date | Days |
|---|---|
| JUN2016 | 30 |
| JUL2016 | 31 |
| AUG2016 | 31 |

The benefit of using a format is that the natural sort order is retained.

---

Using `sashelp.class` as an example, say you wanted to compare the frequency of the first letter of each name. You could use the `substr()` function to find the first letter, and run a `proc freq` on the new variable. Alternatively, you can apply the `$1.` format to the `Name` variable :

```
proc freq data=sashelp.class ;
  table Name ;
  format Name $1. ;
run ;
```

| Name | COUNT |
|---|---|
| A | 7 |
| B | 4 |
| C | 2 |
| etc. | |

## Custom Formats

Custom formats, also known as user defined formats, can be created and used like any other default formats.

```
/*Create new character format for state variables*/
PROC FORMAT;
VALUE $statef      'CA' = 'California'
```

```
                    'MA' = 'Massachusetts'
                    'NY' = 'New York';

/*Once created, you can use your custom format in PROC and DATA steps*/
PROC PRINT DATA=table;
FORMAT state-var $statef.;
RUN;
```

The variable `state-var` will be printed according to the new format. For example, the value `'CA'` will be printed as `'California'`. If a value was not formatted, such as `'CT'`, then that value will be printed as it appears in the data set.

## Using informats to read data

Informats are used to tell SAS how to read in the data and are identified with an `informat` statement.

```
data test;
 infile test.csv;
 informat     id $6.
              date mmddyy10.
              cost comma10.2
 ;
 input @1 id
       @7 date
       @20 cost
 ;
run;
```

Informats and Formats can also be used together to read in the data and write it out in a different format such as with the salary variable below:

```
DATA workers;
  informat first last $16.;
  informat salary 12.1;
  informat birthdate 8.;
  input
    first $
    last $
    birthdate
    salary;
  format salary dollar10.;
datalines;
John Smith 19810505 54998.5
Jane Doe 19950925 45884.5
Frank James 19600222 70000.5
Jamie Love 19630530 292000.5
;
run;
```

Read SAS Formats online: https://riptutorial.com/sas/topic/5010/sas-formats

# Chapter 11: SAS Labels

## Remarks

Labels can be used to describe a variable which helps improve the readability of your outputs. Labels can be permanently created in the `DATA` step or temporarily created in a `PROC` step.

## Examples

### Create Permanent Variable Labels in DATA step

```
data table;
    set table;
    label variable1 = 'label1'
            variable2 = 'label2'
            variable3 = 'label3';
run;
```

Read SAS Labels online: https://riptutorial.com/sas/topic/7877/sas-labels

# Chapter 12: Sending an email with SAS

## Introduction

There are several reasons you might come across for needing email capabilities in SAS. You could be sending an email to notify someone that a process passed/failed, you could be sending an email containing Macro Variables that show how many records have been loaded at the end of your data feed, or maybe you need to send some files that contain reports. Whatever your need is, there are several ways to go about sending emails and files in SAS.

## Parameters

| Tag/Attribute | Value |
|---|---|
| LRECL | This parameter is used to define record length when reading and writing files. I've solved many issues by just setting this to its max value, which is 32767. It's very possible that setting something like this to its max value is less efficient, but at the end of the day it gets the job done for me without any felt performance loss. (the range for LRECL is 1-32767) |

## Examples

### Sending a basic text email with SAS

```
Filename myEmail EMAIL
    Subject = "My Email Subject"
    From    = "myFromAddress@email.com"
    To      = 'toAddress@email.com'
    CC      = 'ccAddress@email.com'
    Type    = 'Text/Plain';


Data _null_; File myEmail;
    PUT "Email content";
    PUT "&recordsCount loaded to your favorite table today!";
RUN;
```

### Attaching an excel file to your SAS email

```
Filename myEmail EMAIL
    Subject = "My Email Subject"
    From    = "myFromAddress@email.com"
    To      = 'toAddress@email.com'
    CC      = 'ccAddress@email.com'
    Type    = 'Text/Plain'
    ATTACH = ("my/excel/file/path/file.extension" content_type="application/vnd.ms-excel"
LRECL= 32767);
```

```
Data _null_; File myEmail;
    PUT "Email contentent";
    PUT "&recordsCount loaded to your favorite table today!";
RUN;
```

## Sending a SAS email with an HTML body

Take note of the email type: Type = 'text/html';

```
Filename myEmail EMAIL
    Subject = "My Email Subject"
    From    = "myFromAddress@email.com"
    To      = 'toAddress@email.com'
    CC      = 'ccAddress@email.com'
    Type    = 'text/html';

Data _null_; File myEmail;
PUT "
<html>
    <head>
        <style>
            table, th, td {
                border: 1px solid black;
                   border-collapse: collapse;
            }
        </style>
    </head>
    <body>
        <p>Here is your email</p>
        <p>Go ahead, organize your data within an HTML table tag here!</p>
        <table>
            <tr>
                <th>
                    column 1
                </th>
                <th>
                    column 2
                </th>
            </tr>
            <tr>
                <td>
                    &countOfRecords1
                </td>
                <td>
                    &countOfRecords2
                </td>
            </tr>
        </table>
    </body>
</html>
";
RUN;
```

It is very possible that after building out an HTML email in SAS, you find the HTML is distorted when you receive the email. This is a result of SAS putting breaks to the next line in the text of your PUT. A break was probably placed right in the middle of one of your tag's text. *Should this happen to you, try moving your HTML tags around. It may not be pretty, but you may have to have some tags share a line to avoid this happening.* This happened to me, and this is exactly how I fixed that issues.

Read Sending an email with SAS online: https://riptutorial.com/sas/topic/9398/sending-an-email-with-sas

# Chapter 13: Using Joins in SAS

## Introduction

Each database is a collection of different tables and each table contains different data in an organized way. While working with data, most of the times information we need is scattered in more than one table. We need joins/merge to get the desired output.

In SAS we use joins while working with `Proc SQL` and use merge while working with `Data step`. We will now talk only about joins inside `Proc SQL`.

## Parameters

| Type of join | Output |
|---|---|
| Proc Sql | SQL procedure inside SAS |
| Create Table | Creates a SAS dataset |
| Select | Selects required variables from respective datasets |
| Where | Specifies particular condition |
| Quit | End the procedure |

## Remarks

As mentioned in the introduction, we can also use `Merge`inside a `data step` which will be discussed under a separate topic. Joins play a very important role to blend and unify data according to the requirement.

## Examples

### Vertical Joining

Vertical join appends dataset B to dataset A providing both of them have similar variables. For example, we have sales for the month of Jan'17 in dataset A and sales for Feb'17 in dataset B. To create a dataset C that has sales of both Jan and Feb we use Vertical Join.

```
PROC SQL;
CREATE TABLE C AS
SELECT *
FROM A
UNION
SELECT *
```

```
FROM B;
QUIT;
```

Now dataset C has observations from both A and B and is appended vertically.

## Inner Join

Inner join creates a dataset that contains records that have matching values from both the tables. For example, we have a dataset A that contains customer information and a dataset B that contains credit card details. To get the credit card details of customers in dataset A, let us create dataset C

```
PROC SQL;
CREATE TABLE C AS
SELECT A.*, B.CC_NUM
FROM CUSTOMER A, CC_DETAILS B
WHERE A.CUSTOMERID=B.CUSTOMERID
QUIT;
```

Dataset C will have only matching observations from both the datasets.

## Left Join

Left join returns all the observations in the left data set regardless of their key values but only observations with matching key values from the right data set. Considering the same example as above,

```
PROC SQL;
CREATE TABLE C AS
SELECT A.*, B.CC_NUMBER, B.START_DATE
FROM CUSTOMER A LEFT JOIN CC_DETAILS B
ON A.CUSTOMERID=B.CUSTOMERID
QUIT;
```

Dataset C contains all the values from the left table, plus matched values from the right table or missing values in the case of no match.

## Right join

Like left join, right join selects all the observations from the right dataset and the matched records from the left table.

```
PROC SQL;
CREATE TABLE C AS
SELECT A.*, B.CC_NUMBER, B.START_DATE
FROM CUSTOMER A RIGHT JOIN CC_DETAILS B
ON A.CUSTOMERID=B.CUSTOMERID
QUIT;
```

Dataset C contains all the values from the right table, plus matched values from the left table or missing values in the case of no match.

## Full Join

Full join selects all the observations from both data sets but there are missing values where the key value in each observation is found in one table only.

```
PROC SQL;
CREATE TABLE C AS
SELECT A.*, B.CC_NUMBER, B.START_DATE
FROM CUSTOMER A FULL JOIN CC_DETAILS B
ON A.CUSTOMERID=B.CUSTOMERID
QUIT;
```

Dataset C will contain all records from both the tables and fill in `.` for missing matches on either side.

Read Using Joins in SAS online: https://riptutorial.com/sas/topic/9900/using-joins-in-sas

# Chapter 14: Variable Length

## Syntax

- LENGTH variable(s) <$>length;

## Parameters

| Parameter | Details |
|---|---|
| variable(s) | variable(s) you wish to assign a length to |
| $ | optional parameter that specifies if your variable is a character variable |
| length | integer that specifies the length of the variable |

## Examples

### Assigning length to a character variable

```
data table;
set table;
length state_full $8;
if state = 'KS' then state_full = 'Kansas';
else if state = 'CO' then state_full = 'Colorado';
else state_full = 'Other';
run;
```

Read Variable Length online: https://riptutorial.com/sas/topic/7883/variable-length

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with sas | Bendy, brusso, Chris J, Community, dcudonk, fl0r3k, Jay Stevens, Joe |
| 2 | Copy a file, byte for byte | Joshua Schlichting |
| 3 | Creating Macro Variables | Joshua Schlichting |
| 4 | data step | zuluk |
| 5 | DO Loop | heyydrien, zuluk |
| 6 | Informats in SAS | Praneeth Rachumallu, Robert Penridge |
| 7 | Proc SQL | Altons, D. O., Jay Stevens |
| 8 | Reading Data | GForce, heyydrien, Joshua Schlichting |
| 9 | Resolving Macro Variables in quotes within PROC SQL Pass-throughs | Joshua Schlichting |
| 10 | SAS Formats | Chris J, GForce, heyydrien, Robert Penridge |
| 11 | SAS Labels | heyydrien |
| 12 | Sending an email with SAS | Joshua Schlichting |
| 13 | Using Joins in SAS | Praneeth Rachumallu |
| 14 | Variable Length | heyydrien |