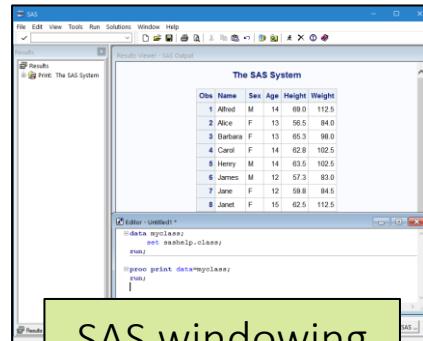
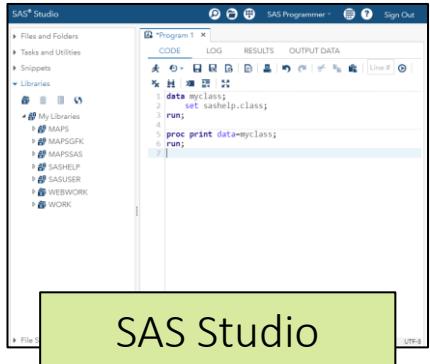


# SAS Lesson 01

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# SAS Programming Interfaces



All these interfaces have the basic tools that you need for programming.

# SAS Programming Interfaces

write  
and  
submit  
code

```
data myclass;  
  set sashelp.class;  
run;  
  
proc print data=myclass;  
run;
```

Editor

```
1  data myclass;  
2  set sashelp.class;  
3  run;
```

view  
messages  
from SAS

NOTE: There were 19 observations read from the data set SASHHELP.CLASS.

NOTE: The data set WORK.MYCLASS has 19 observations and 5 variables.

NOTE: DATA statement used:

real time 0.01 seconds  
cpu time 0.00 seconds

4

```
5  proc print data=myclass;
```

NOTE: Writing HTML Body file: sashtml.htm  
6 run;

NOTE: There were 19 observations read from the data set WORK.MYCLASS.

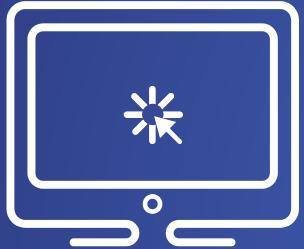
Log

view  
results

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0

Janet	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5

Results and  
Output Data



# PC SAS and SAS on Demand

This demonstration provides an overview of the two programming interfaces that will be used in STAT604.

# Setting up the Course Files

- Only required for full (PC) SAS installation
- Download zip file:

<http://support.sas.com/content/dam/SAS/support/en/books/data/base-guide-practice-data.zip>

- Unzip file in accessible location
- PC SAS
  - C:\Users\myname\Documents\STAT604Data\
- Browse to cert folder under the new unzipped folder
  - Use explorer for PC SAS
- Copy folder path

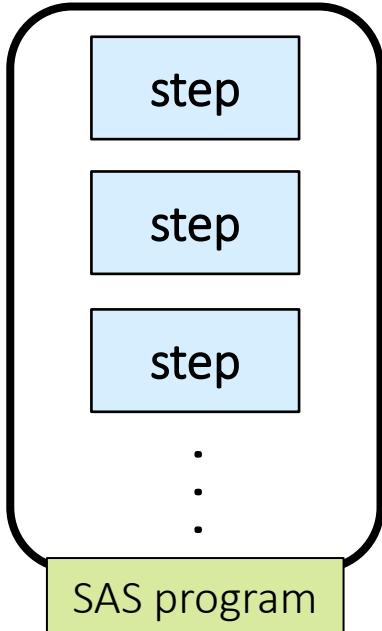
# Setting up the Course Files

- Open the program ..cert\cr8data.sas
- Use paste to replace path on the %let path= value
- Run the program 

# Essentials

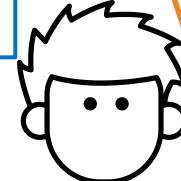
Understanding SAS Syntax

# SAS Program Structure



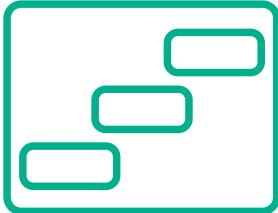
```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

A SAS program  
consists of a  
sequence of steps.

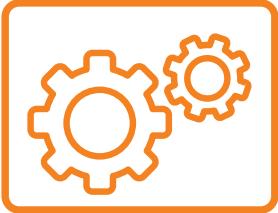


# SAS Program Structure

DATA step

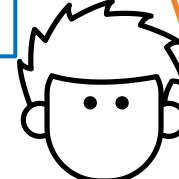


PROC step



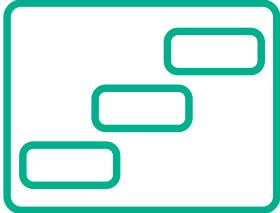
```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

A program can be  
any combination  
of DATA and PROC  
(procedure) steps



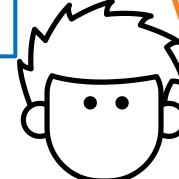
# SAS Program Structure

## DATA step



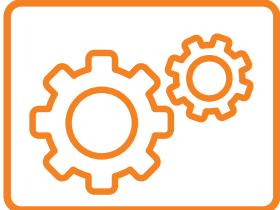
```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

DATA steps typically read, process, or create data.



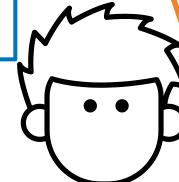
# SAS Program Structure

PROC step



```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

PROC steps typically report, manage, or analyze data.



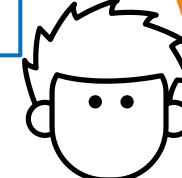
# SAS Program Structure

Steps begin with either DATA or PROC.

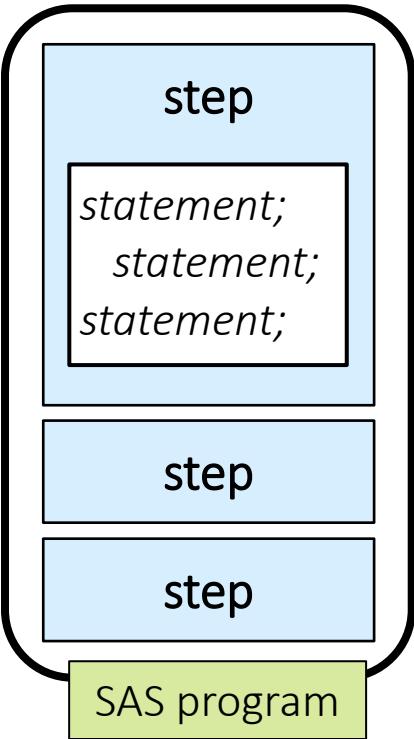
```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

Steps end with RUN.  
Some PROCs end with QUIT.

This program has three steps.

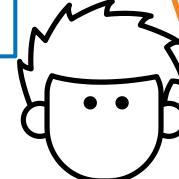


# SAS Program Structure



```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

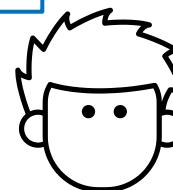
A step is a sequence of SAS statements.



# SAS Statement Syntax

```
data myclass;  
  set sashelp.class;  
  heightcm=height*2.54;  
run;  
  
proc print data=myclass;  
run;  
  
proc means data=myclass;  
  var age heightcm;  
run;
```

Most statements begin with a keyword, and all statements end with a semicolon.



# Global Statements

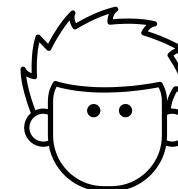
OPTIONS ... ;

TITLE ... ;

LIBNAME ... ;

Run is  
not a global  
statement

Global statements  
are typically  
outside of steps  
and do not need a  
RUN statement.



# Activity

```
data mycars;  
  set sashelp.cars;  
  AvgMPG=mean(mpg_city, mpg_highway);  
run;  
  
title "Cars with Average MPG Over 35";  
proc print data=mycars;  
  var make model type avgmpg;  
  where AvgMPG > 35;  
run;  
  
title "Average MPG by Car Type";  
proc means data=mycars mean min max maxdec=1;  
  var avgmpg;  
  class type;  
run;  
  
title;
```

Steps? 3

Statements in PROC PRINT? 4

Global Statements? 3

3

→ fifth

# SAS Program Syntax: Format

These are  
the same  
to SAS.

```
data myclass; set sashelp.class; run;  
proc print data=myclass; run;
```

```
data myclass;  
    set sashelp.class;  
run;  
  
proc print data=myclass;  
run;
```

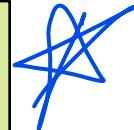
Formatting makes  
your code easier  
to read and  
understand.



# SAS Program Syntax: Case

```
data under13;  
  set sashelp.class;  
  where AGE<13;  
  drop height weight;  
run;
```

Unquoted values can  
be in any case.



# SAS Program Syntax: Comments

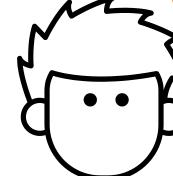
```
/* students under 13 yo */  
  
data under13;  
    set sashelp.class;  
    where Age<13;  
    *drop Height Weight;  
run;
```

comments out everything between /\* and \*/

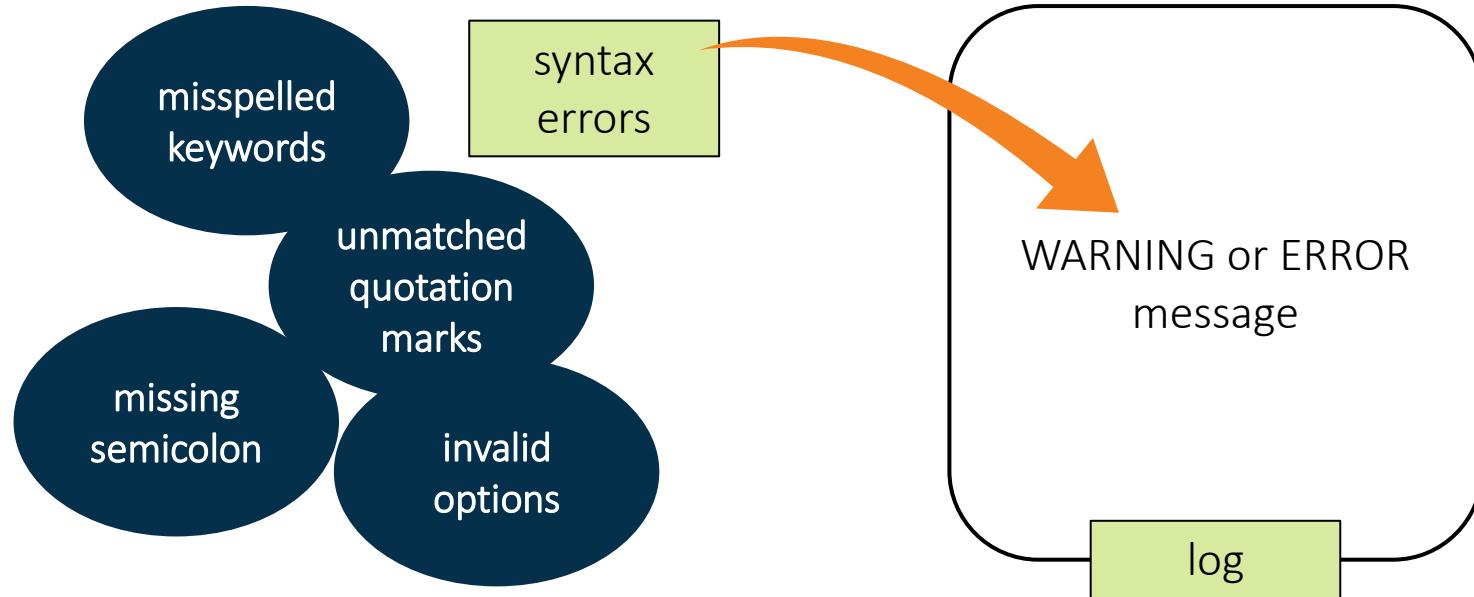
comments out a single statement ending in a semicolon

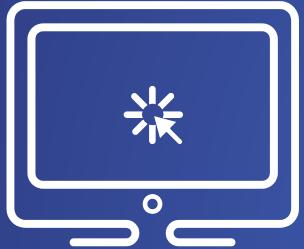


Comments are ignored when a program executes.



# Finding and Resolving Syntax Errors





# Finding and Resolving Syntax Errors Using comments

This demonstration illustrates finding and resolving common syntax errors. It also demonstrates comment types.

\ehs\ehs02.sas

# Referencing SAS Files

SAS Libraries Overview – Prep Guide Chapter 2

# SAS Data Libraries

A SAS *data library* is a collection of SAS files or tables that are recognized as a unit by SAS. (Excludes raw data files.)

Directory-based System

A SAS data library is a directory.

Windows Example: `c:\users\userid\cert`

UNIX Example: `/users/userid/cert`

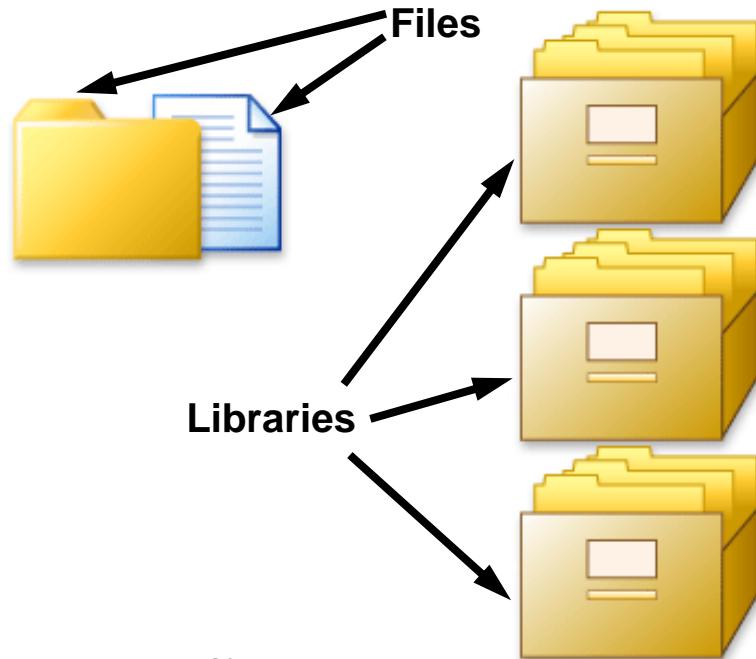
z/OS (OS/390)

A SAS data library is an operating system file.

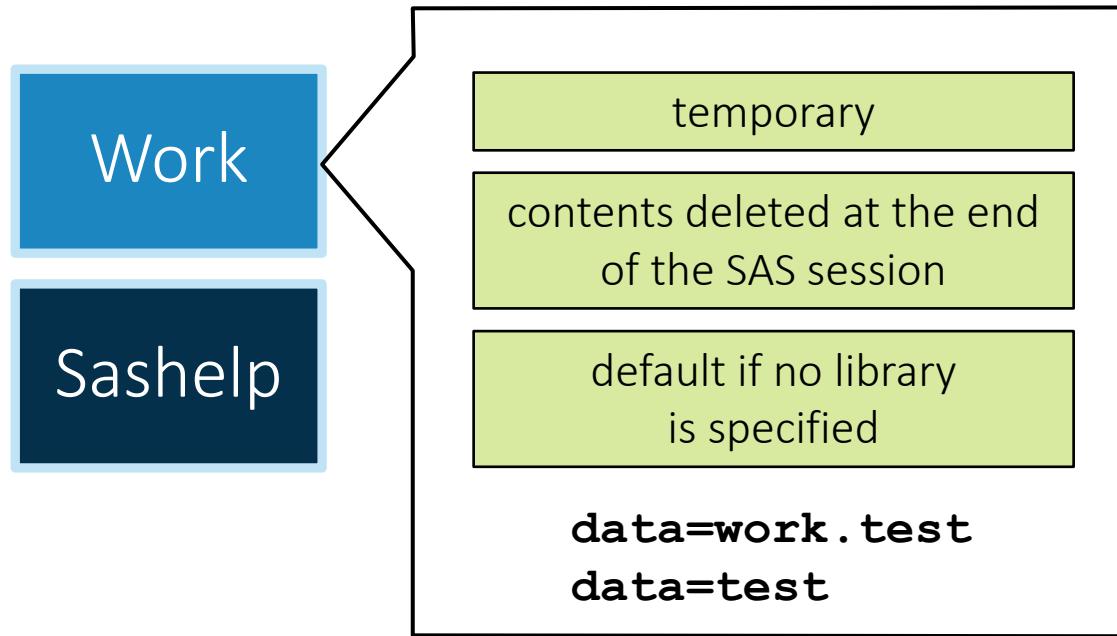
z/OS (OS/390) Example: `userid.workshop.sasdata`

# SAS Data Libraries

You can think of a SAS data library as a drawer in a filing cabinet and a SAS data set as one of the file folders in the drawer.



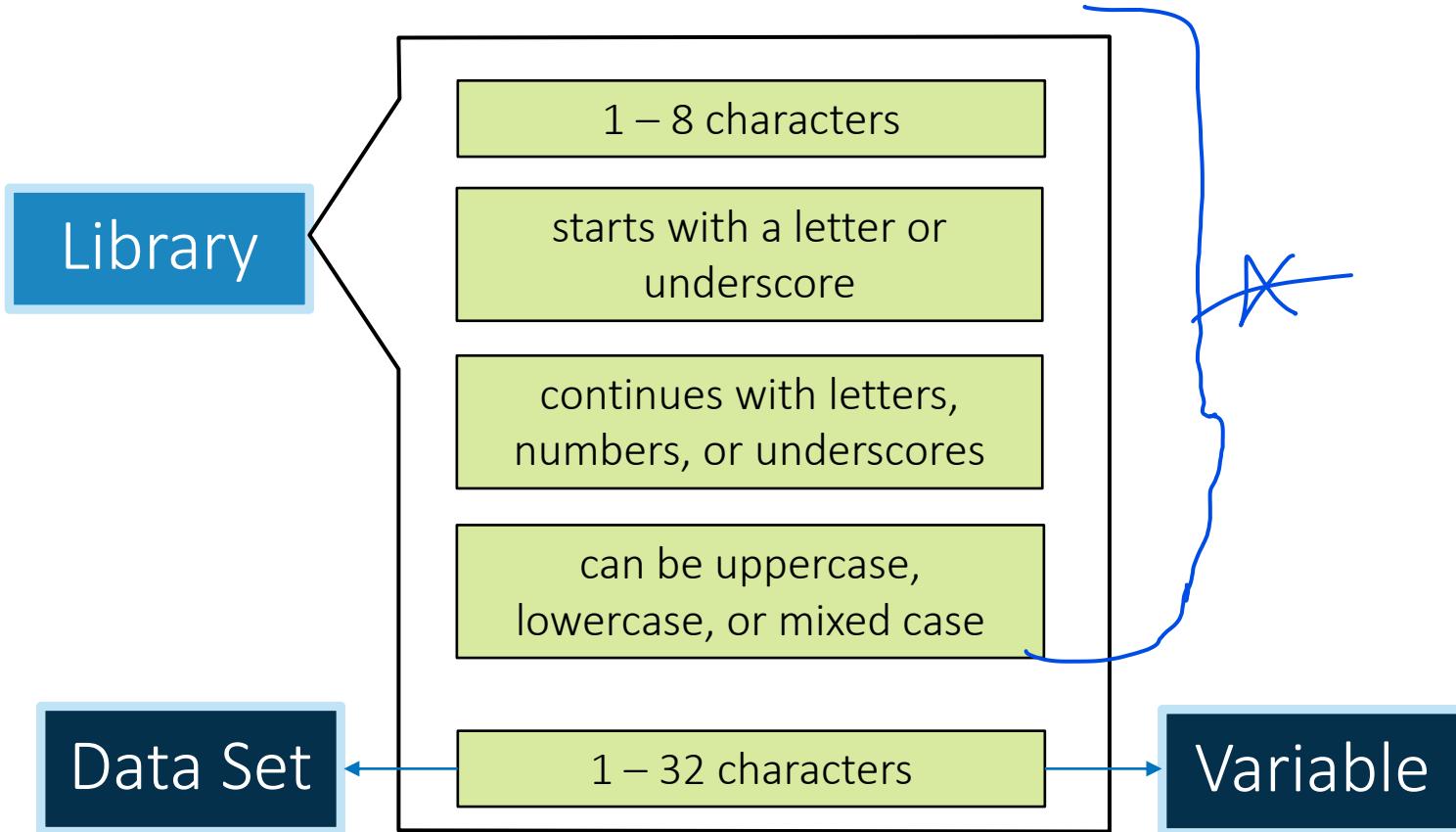
# Automatic SAS Libraries



# Automatic SAS Libraries



# Default SAS Naming Rules



# Changing the Default Behavior

```
options option1=optionvalue option2;
```

system option  
with value

additional options  
(may or may not use  
a value)

```
proc options;  
run;
```

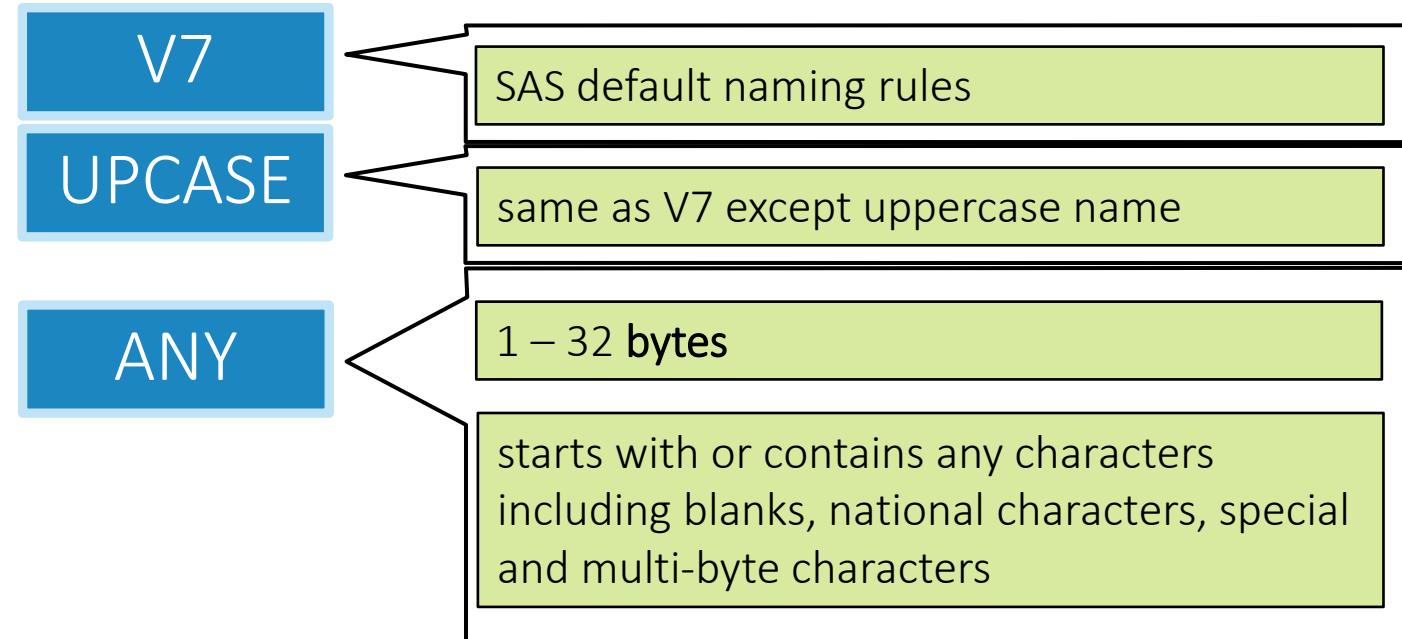
write current  
options to SAS log

System  
options are  
global!



# Changing the Default Behavior for Variables

```
options validvarname=V7|UPCASE|ANY;
```



# Changing the Default Behavior for Tables

```
options validmemname=COMPATIBLE|EXTEND;
```

COMPATIBLE

SAS default naming rules

EXTEND

1 – 32 bytes

includes national characters

special characters except / \ \* ? " < > | : -

cannot begin with blank or period

# Lesson Quiz



1. How many steps does this program contain?

- a. one
- b. two
- c. four
- d. eight

```
data national;  
    set sashelp.baseball;  
    BatAvg=nHits/nAtBat;  
run;  
  
proc contents data=national;  
run;  
  
proc print data=national;  
run;  
  
proc means data=national;  
    var BatAvg;  
run;
```

1. How many steps does this program contain?

- a. one
- b. two
- c. four
- d. eight

```
data national;  
    set sashelp.baseball;  
    BatAvg=nHits/nAtBat;  
run;  
  
proc contents data=national;  
run;  
  
proc print data=national;  
run;  
  
proc means data=national;  
    var BatAvg;  
run;
```

2. Running a SAS program can create which of the following?
- a. log
  - b. output data
  - c. results
  - d. all of the above

2. Running a SAS program can create which of the following?

- a. log
- b. output data
- c. results
- d. all of the above

3. Which of the following is a SAS syntax requirement?
- a. Begin each statement in column one.
  - b. Put only one statement on each line.
  - c. Separate each step with a line space.
  - d. End each statement with a semicolon.

3. Which of the following is a SAS syntax requirement?

- a. Begin each statement in column one.
- b. Put only one statement on each line.
- c. Separate each step with a line space.
- d. End each statement with a semicolon.

4. How many statements does this program contain?

- a. five
- b. six
- c. seven
- d. eight

```
*Create a cars report;  
  
title "European Cars Priced Over 30K";  
footnote "Internal Use Only";  
  
proc print data=sashelp.cars;  
    where Origin='Europe'  
          and MSRP>30000;  
    var Make Model Type  
        Mpg_City Mpg_Highway;  
run;
```

4. How many statements does this program contain?

- a. five
- b. six
- c. seven
- d. eight

```
*Create a cars report;  
  
title "European Cars Priced Over 30K";  
footnote "Internal Use Only";  
  
proc print data=sashelp.cars;  
    where Origin='Europe'  
          and MSRP>30000;  
    var Make Model Type  
        Mpg_City Mpg_Highway;  
run;
```

5. Which of the following steps is typically used to generate reports and graphs?
- a. DATA
  - b. PROC
  - c. REPORT
  - d. RUN

5. Which of the following steps is typically used to generate reports and graphs?
- a. DATA
  - b. PROC
  - c. REPORT
  - d. RUN

6. Does this comment contain syntax errors?

```
/*
Report created for budget
presentation; revised October 15.
*/
proc print data=work.newloan;
run;
```

- a. No. The comment is correctly specified.
- b. Yes. Every comment line must end with a semicolon.
- c. Yes. The comment is on more than one line.
- d. Yes. There is a semicolon in the middle of the comment.

6. Does this comment contain syntax errors?

```
/*
Report created for budget
presentation; revised October 15.
*/
proc print data=work.newloan;
run;
```

- a. No. The comment is correctly specified.
- b. Yes. Every comment line must end with a semicolon.
- c. Yes. The comment is on more than one line.
- d. Yes. There is a semicolon in the middle of the comment.

7. What result would you expect from submitting this step?

```
proc print data=work.newsalesemps  
run;
```

- a. a report of the **work.newsalesemps** data set
- b. an error message in the log
- c. the creation of a table named **work.newsalesemps**

7. What result would you expect from submitting this step?

```
proc print data=work.newsalesemps  
run;
```

- a. a report of the `work.newsalesemps` data set
- b. an error message in the log
- c. the creation of a table named `work.newsalesemps`

8. What happens if you submit the following program?

```
proc print data=work.newsalesemps;  
run;
```

- a. SAS does not execute the step.
- b. SAS assumes that PROC is misspelled and executes the step.

8. What happens if you submit the following program?

```
proc print data=work.newsalesemps;  
run;
```

- a. SAS does not execute the step.
- b. SAS assumes that PROC is misspelled and executes the step.

9. This program contains a syntax error because **National** is in different cases.

```
data national;
  set sashelp.baseball;
  BatAvg=nHits/nAtBat;
run;

proc means data=NATIONAL;
  var BatAvg;
run;
```

- a. True
- b. False

9. This program contains a syntax error because **National** is in different cases.

```
data national;
  set sashelp.baseball;
  BatAvg=nHits/nAtBat;
run;

proc means data=NATIONAL;
  var BatAvg;
run;
```

- a. True
- b. False

10. Which of the following is not a SAS programming interface?
- a. SAS Enterprise Guide
  - b. SAS Manager
  - c. SAS Studio
  - d. SAS windowing environment

10. Which of the following is not a SAS programming interface?
- a. SAS Enterprise Guide
  - b. SAS Manager
  - c. SAS Studio
  - d. SAS windowing environment

# SAS Lesson 02

10(7)21

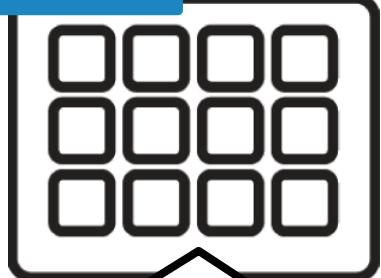
Portions Copyright © 2018 SAS Institute Inc.,  
Cary, NC, USA. All rights reserved. Reproduced  
with permission of SAS Institute Inc., Cary, NC,  
USA. SAS Institute Inc. makes no warranties  
with respect to these materials and disclaims  
all liability therefor.

# Accessing Data

Understanding SAS Data

# Types of Data

Structured data



has meta Data  
Data about the  
rows / columns  
size of file.

Unstructured data



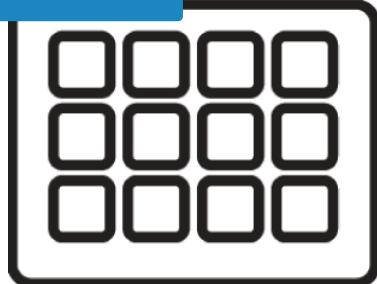
includes defined rows and columns

many types able to be read by SAS

SAS, Oracle, Teradata, Microsoft Excel,  
Hadoop, Versa tables, and others

# Types of Data

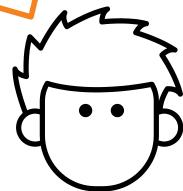
Structured data



Unstructured data



Metadata makes the difference!

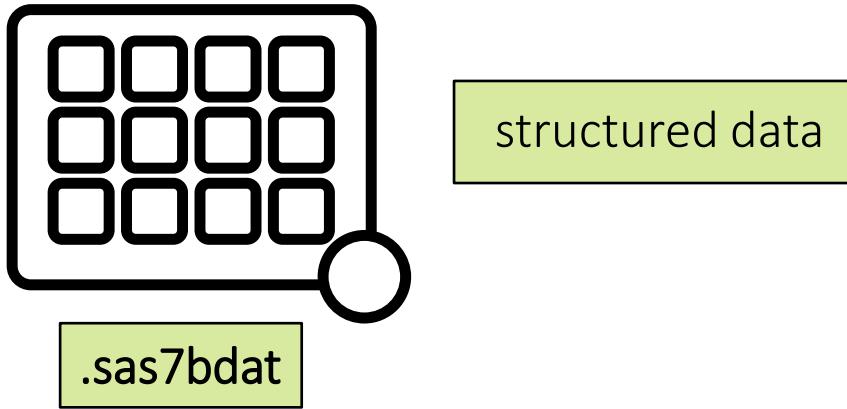


includes data, but not defined columns

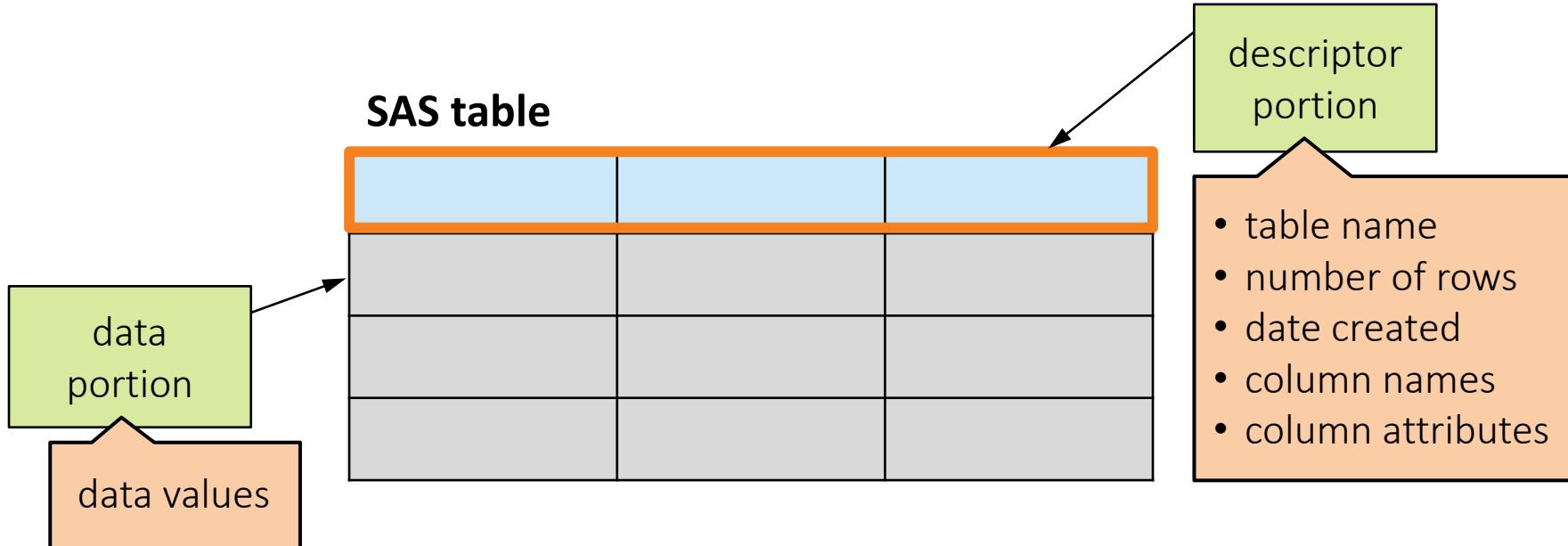
must be imported into SAS

text, delimited, JSON, weblogs, and other files

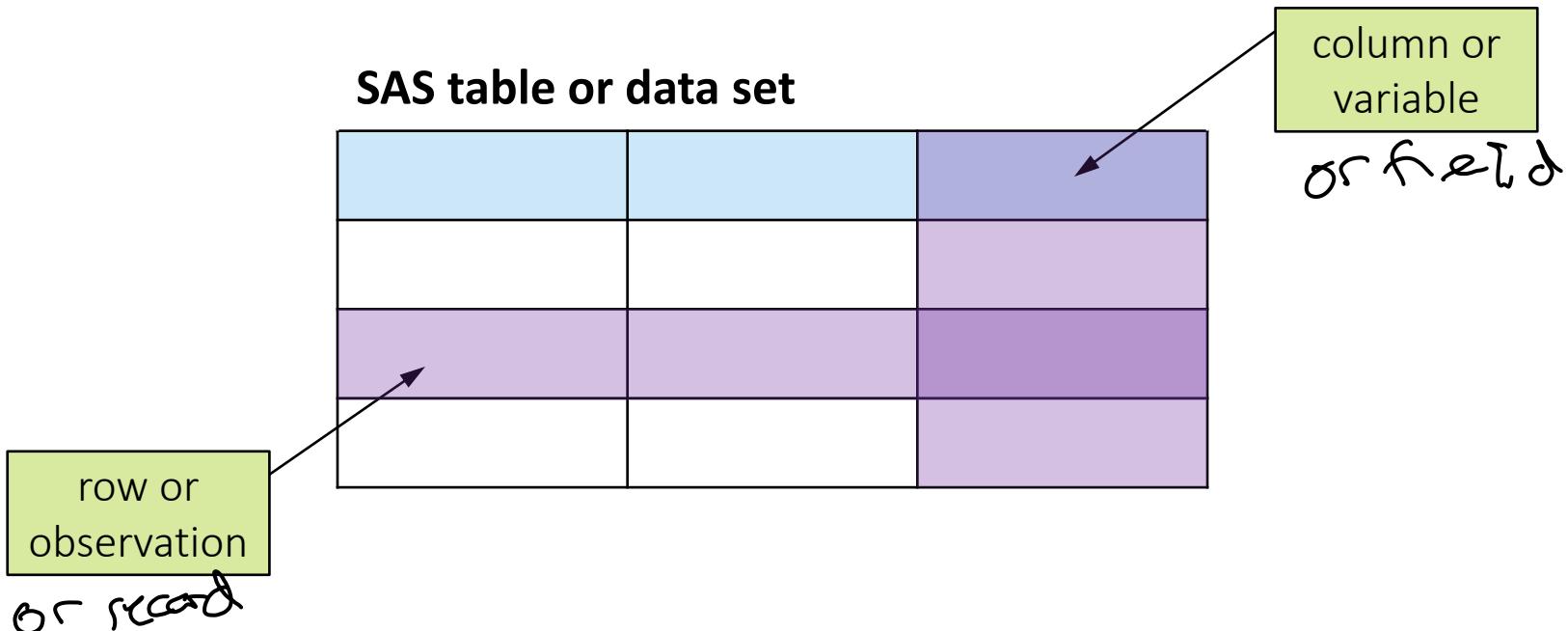
# What Is a SAS Table?



# What Is a SAS Table?



# SAS Terminology



# Required Column Attributes for SAS Tables

Name

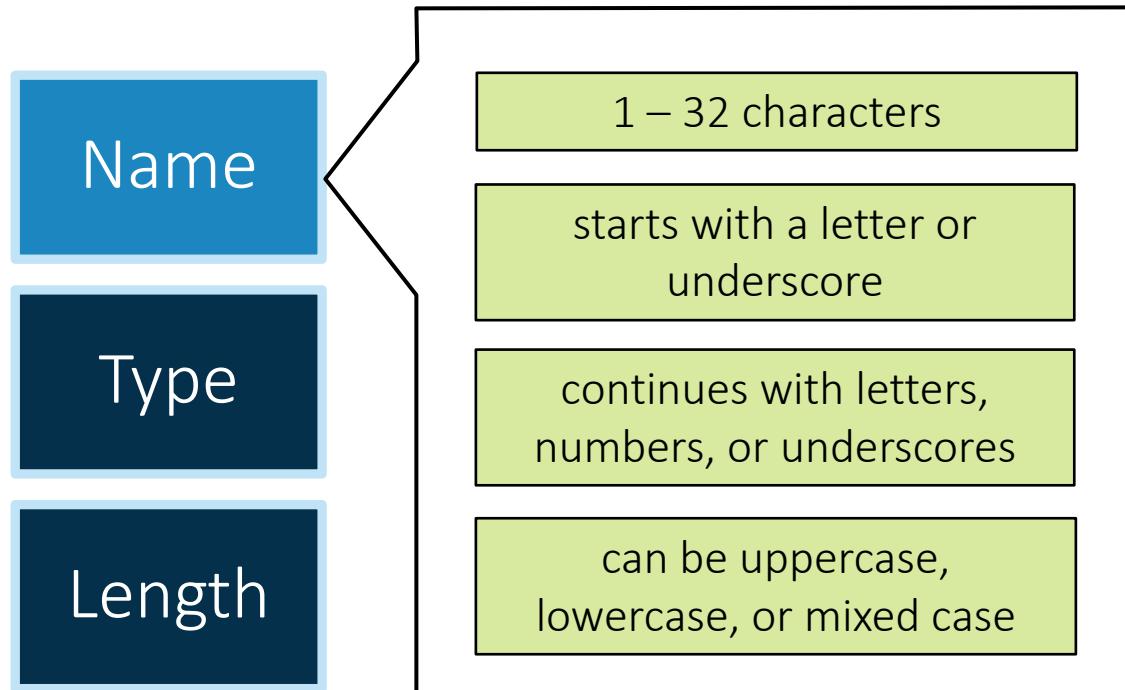
Type

Length



In SAS, all columns must have a name, type, and length.

# Required Column Attributes: Name



X NO  
spac(a)  
charachz  
on col / Name

# Multiple Answer Question

Which column names are valid? (Select all that apply.)

- a. month6
- b. 6month
- c. month#6
- d. month 6
- e. month\_6
- f. Month6

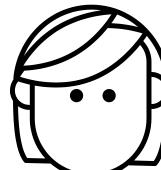
# Multiple Answer Question – Correct Answers

Which column names are valid? (Select all that apply.)

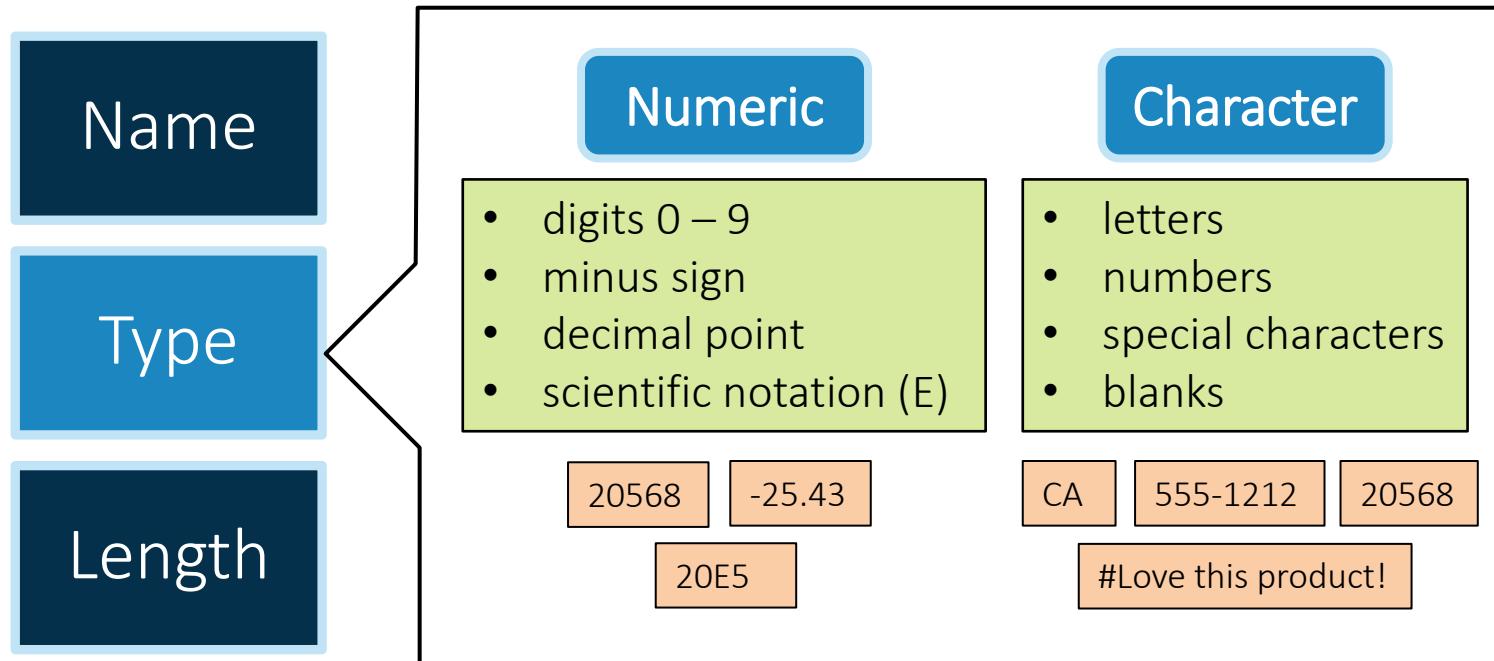
- a. month6
- b. 6month
- c. month#6
- d. month 6
- e. month\_6
- f. Month6

Same name, isn't it?  
Col sas sample  
Not sensible

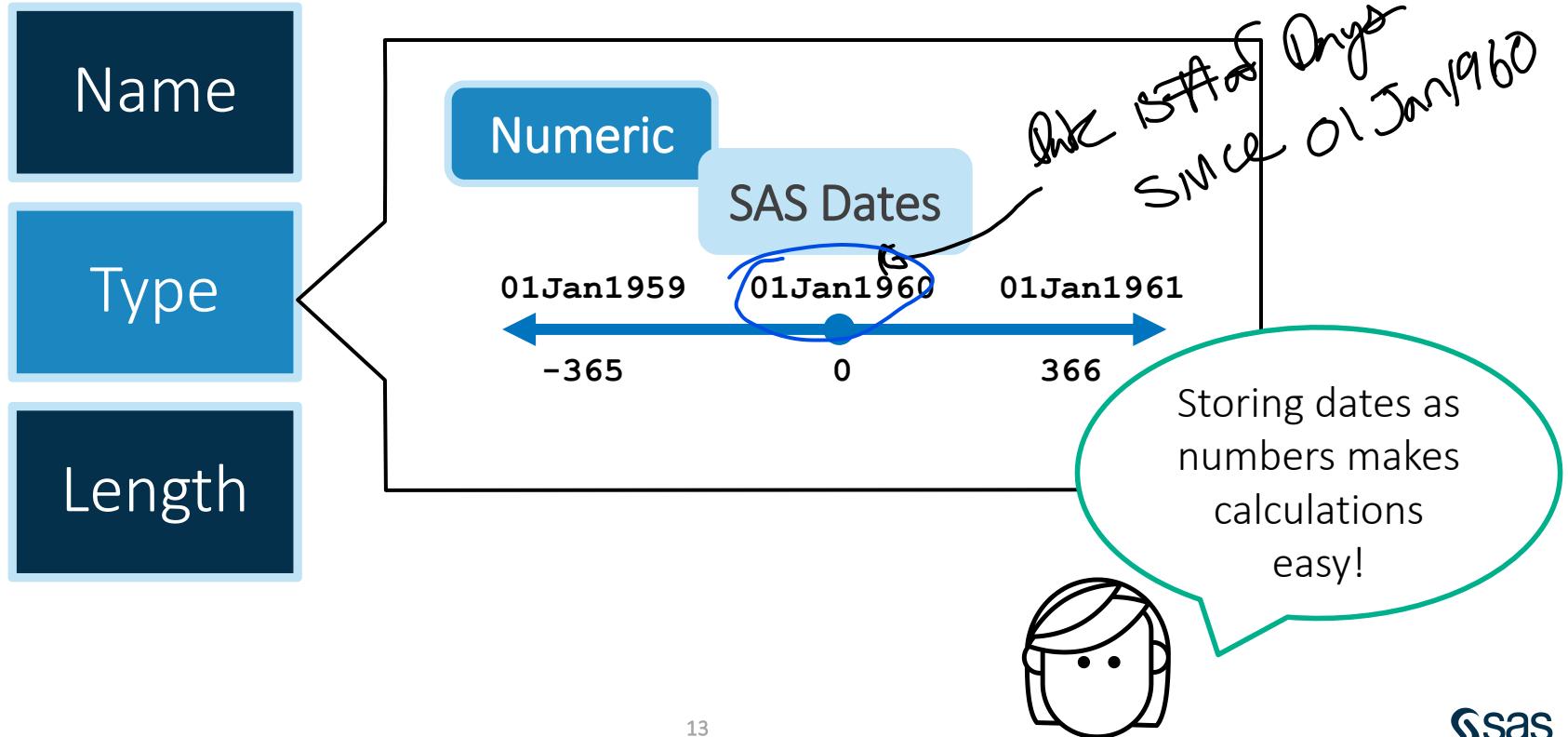
Month6 and month6  
are actually the  
same column name.



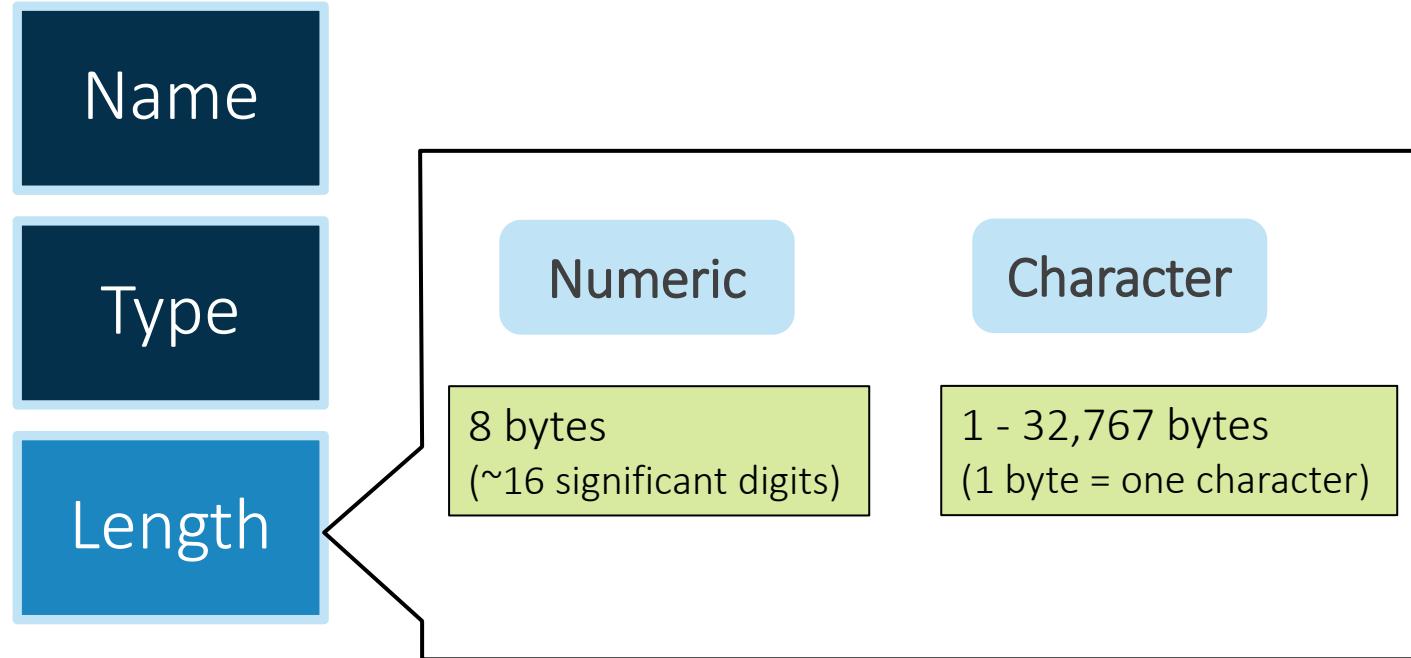
# Required Column Attributes: Type



# Required Column Attributes: Type



# Required Column Attributes: Length



# Activity

How are missing character and numeric values represented in the data shown below?

→ blank for char  
period for numeric

	Season	Name	Basin	Type	MaxWindMPH	MinPressure
1	1980		na	TS	35	.
2	1980		SP	NR	.	998
3	1980	AGATHA	EP	TS	115	.
4	1980	ALBINE	SI	ET	.	.
5	1980	ALEX	WP	TS	40	998
6	1980	ALLEN	NA	TS	190	899
7	1980	AMY	SI	NR	132	915
8	1980	BERENICE	SI	TS	.	.
9	1980	BETTY	WP	ET	115	925
10	1980	BLAS	EP	TS	58	.
11	1980	BONNIE	NA	ET	98	975
12	1980	BRIAN	SI	NR	115	930
13	1980	CARMEN	WP	TS	69	985
14	1980	CARY	WP	TS	52	996
15	1980	CELIA	EP	TS	75	.
16	1980	CHARLEY	NA	TS	81	989

# Activity – Correct Answer

How are missing character and numeric values represented in the data?

	Season	Name	Basin	Type	MaxWindMPH
1	1980	NA	TS		35
2	1980	SP	NR		.
3	1980	AGATHA	EP	TS	115
4	1980	ALBINE	SI	ET	.

blank for  
character

period for  
numeric

# Question

Examine the length of the **Basin** column. Could *East Pacific* be properly stored as a data value in the **Basin** column?

- Yes
- No

Column Name	Type	Length	Format
Season	Num...	8	
Name	Text	57	
Basin	Text	2	
Type	Text	2	

# Question – Correct Answer

Examine the length of the **Basin** column. Could *East Pacific* be properly stored as a data value in the **Basin** column?

- Yes
- No

**Basin** is two bytes,  
so *East Pacific* would  
be truncated, and  
the value would be  
*Ea.*

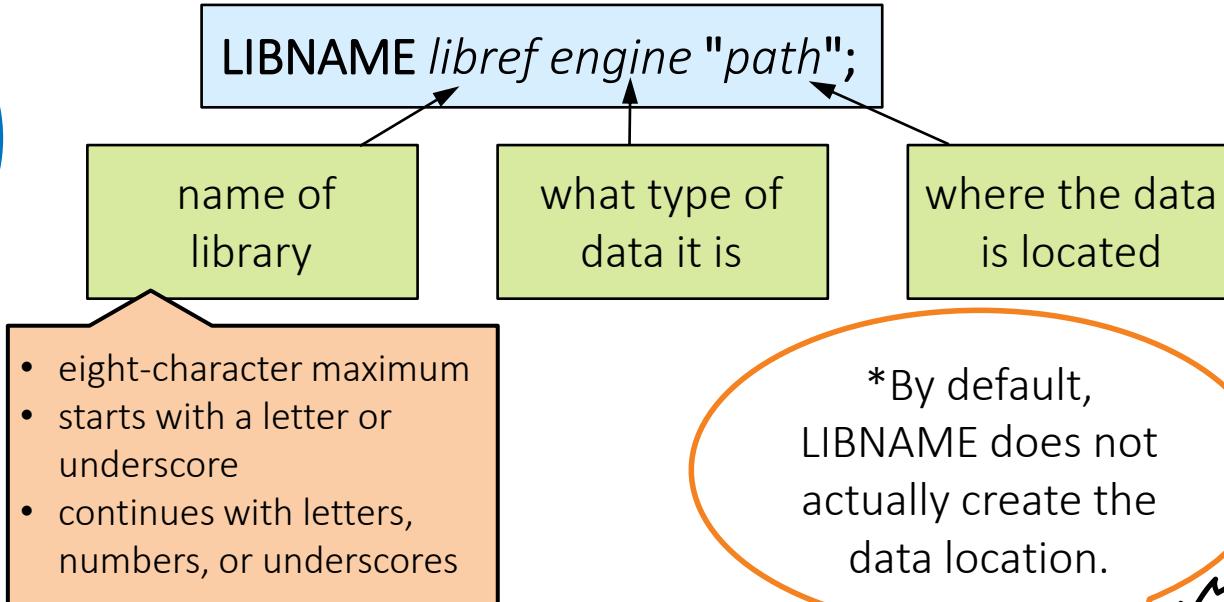


# Accessing Data

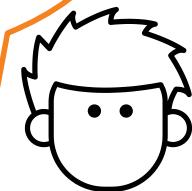
Accessing Data through Libraries – Chapter 3

# Using a Library to Read SAS Files

create  
the  
library\*



\*By default,  
LIBNAME does not  
actually create the  
data location.



# Using a Library to Read SAS Files

create  
the  
library

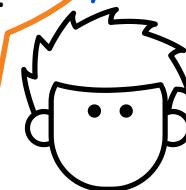
```
libname cert base "c:\users\userid\cert";
```

library name  
cert

Base SAS tables  
(optional)

data located in  
cert folder

LIBNAME is a  
global statement  
and does not need  
a RUN statement.

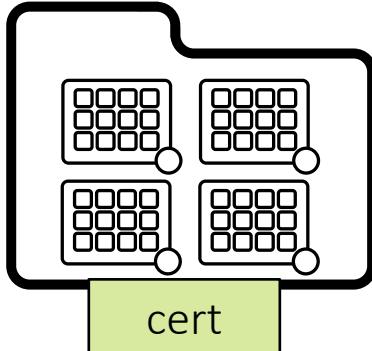


# Using a Library to Read SAS Files

create  
the  
library

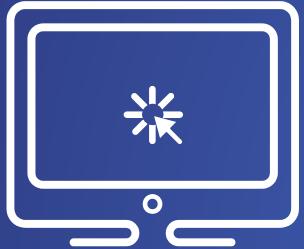
```
libname cert base "c:\users\userid\cert ";
```

```
libname cert "c:\users\userid\cert";
```



The Base SAS engine is the default, so these two statements are the same.





# Assigning a SAS Libref

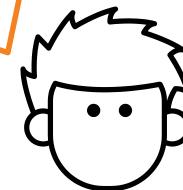
This demonstration/activity will assign a libref to the practice data that accompanies the certification prep guide.

# Activity Question

Why are the Excel and text files in the **cert** folder not included in the library?

Name	Size	Type
Addresses	128.0KB	Table
Admit	128.0KB	Table
Admitjune	128.0KB	Table
Agencyemp	128.0KB	Table
Amounts	128.0KB	Table
Aprbills	128.0KB	Table
Before	128.0KB	Table
Bookcase	128.0KB	Table
Cars	128.0KB	Table
Choices	128.0KB	Table
Class	128.0KB	Table
Clients	128.0KB	Table
Cltrials	128.0KB	Table
Credit	128.0KB	Table
Dates	128.0KB	Table

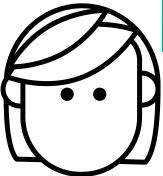
The CERT library uses  
the BASE engine,  
so it reads only  
Base SAS tables.



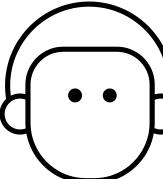
# Viewing Table and Column Attributes

```
PROC CONTENTS DATA=data-set <varnum>;  
RUN;
```

```
proc contents data=cert.class;  
run;
```



PROC CONTENTS creates a report about the descriptor portion of the data.



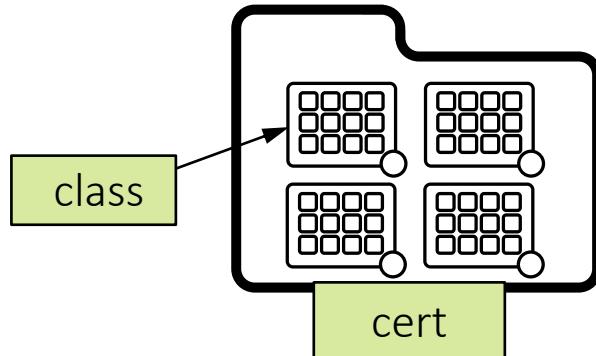
The varnum option lists variables in the order created.

# Using a Library with PROC CONTENTS

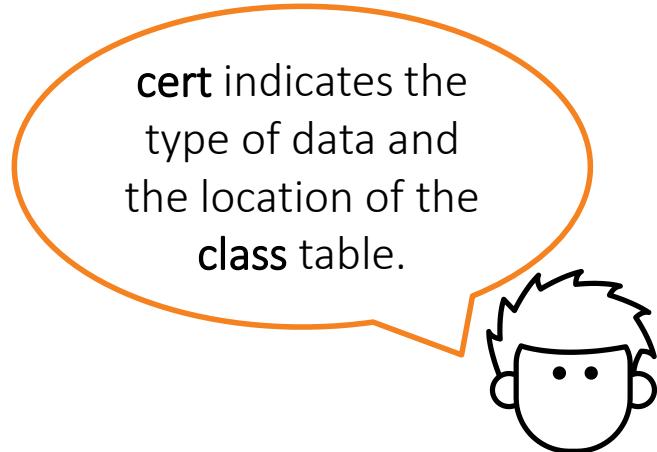


*libref.table-name*

```
proc contents data=cert.class;  
run;
```



**cert** indicates the type of data and the location of the **class** table.

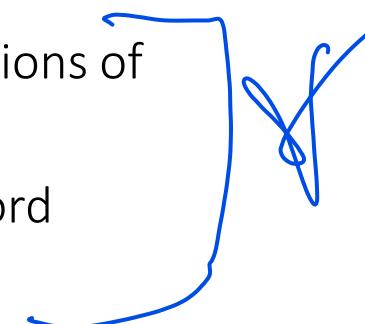




# Browsing a SAS Data Library

The CONTENTS procedure with the `_ALL_` keyword produces a list of all the SAS files in the data library.

```
PROC CONTENTS DATA=libref._ALL_ NODS;  
RUN;
```

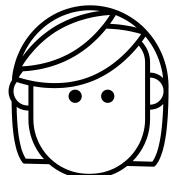
- The NODS option suppresses the descriptor portions of the data sets.
  - NODS is only used in conjunction with the keyword `_ALL_`.
- 

# Viewing Table Data

```
PROC PRINT DATA=data-set <options>;  
RUN;
```

```
proc print data=cert.class;  
run;
```

PROC PRINT creates a report of all rows and columns by default.





# Exploring Automatic SAS Libraries

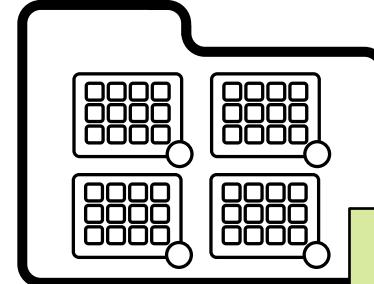
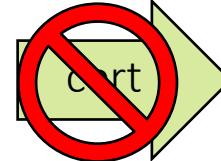
This demonstration illustrates using the **Contents Procedure** to explore automatic and user defined libraries.

# Using a Library to Read SAS Files

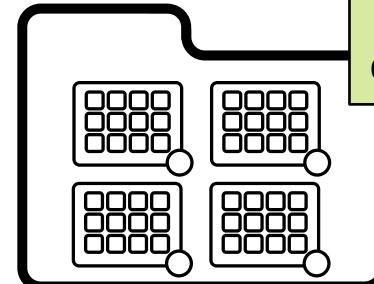
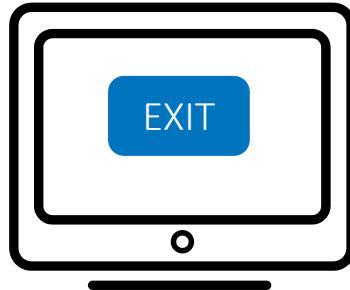


```
libname cert clear;
```

→ Delete library reference → Delete library



Your  
data is  
not  
deleted.



# Referencing SAS Files

SAS ACCESS Libraries – Prep Guide Page 49

# Using a Library to Read Excel Files



```
LIBNAME libref XLSX "path/file-name.xlsx";
```

name of  
the library

The XLSX engine  
reads Excel files.

The path must include  
the complete file  
name and extension.

```
libname certxl xlsx "c:\users\userid\cert\exercise.xlsx";
```

The XLSX engine requires a license  
for SAS/ACCESS Interface to PC Files.

# Using a Library to Read Excel Files

	A	B	C
1	First Name	Last Name	Days Employed
2	Brad	Majors	136
3	Janet	Weiss	
4	Everette	Scott	
5	Frank	Furter	

	⚠ First_Name	⚠ Last_Name	⌚ Days_Employed
1	Brad	Majors	136
2	Janet	Weiss	136
3	Everette	Scott	89
4	Frank	Furter	160

`OPTIONS VALIDVARNAME=V7;`

forces table and column names to follow SAS naming conventions

`LIBNAME libref CLEAR;`

clears the connection to the Excel file

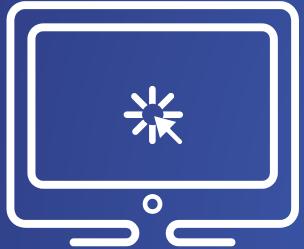
# Using a Library to Read Excel Files

```
libname certxl xlsx "c:\users\userid\cert\exercise.xlsx";
```

```
proc contents data=certxl.ActivityLevels;  
run;
```

```
libname certxl clear;
```

name of the worksheet  
that you want to read



# Using a Library to Read Excel Files

This demonstration illustrates creating a library to connect to an Excel workbook.

# The SAS/ACCESS LIBNAME Statement

The *SAS/ACCESS LIBNAME statement* assigns a library reference name (libref) to a relational database.

General form of the SAS/ACCESS LIBNAME statement:

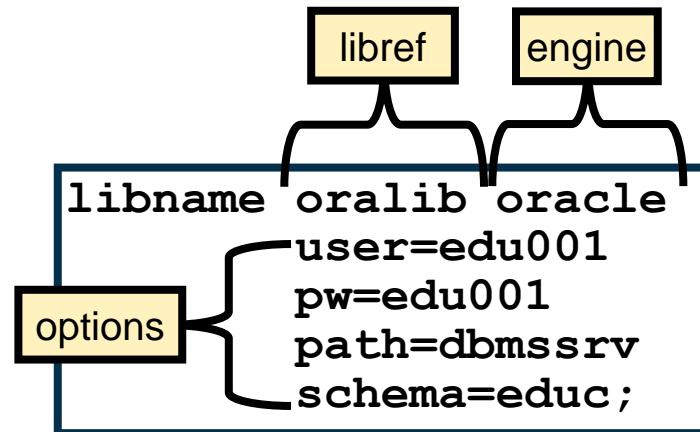
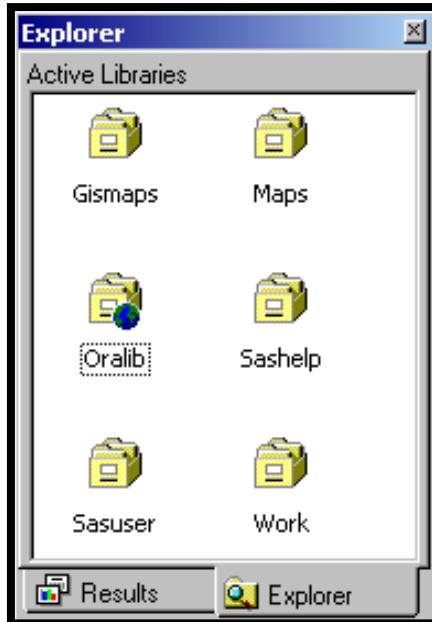
```
LIBNAME libref engine-name <SAS/ACCESS-options>;
```

After a database is associated with a libref, you can use a SAS two-level name to specify any table in the database and then work with the table as you would with a SAS data set.



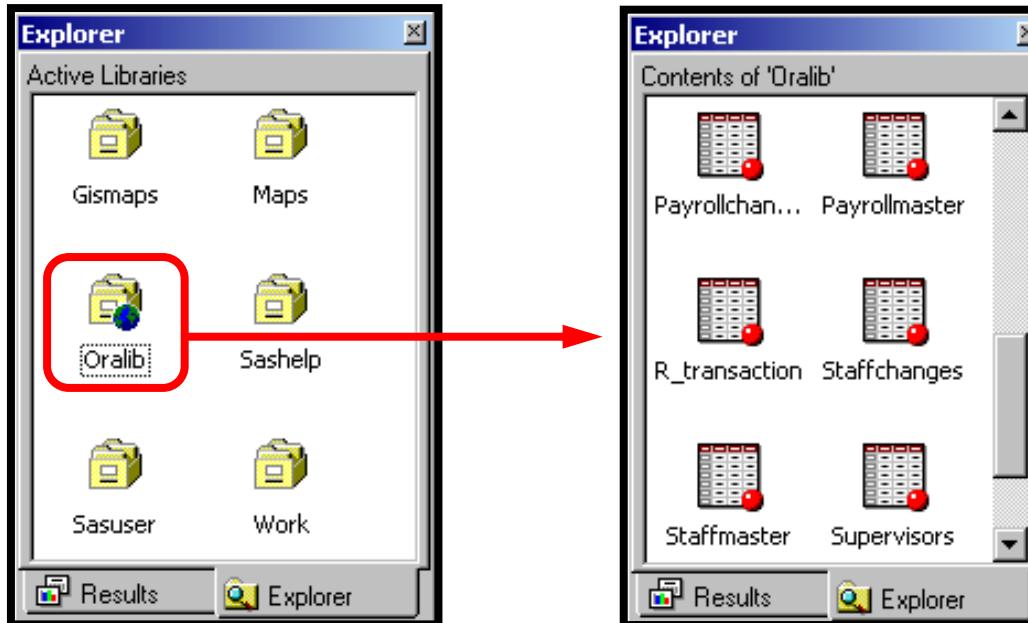
# Oracle Example

This example uses the LIBNAME statement as supported in the SAS/ACCESS interface to Oracle.



# Oracle Example

Any table in this Oracle database can be referenced using a SAS two-level name.



# Oracle Example

```
libname oralib oracle  
      user=edu001 pw=edu001  
      path=dbmssrv schema=educ;  
  
proc print data=oralib.supervisors;  
run;  
  
data work.staffpay;  
    merge oralib.staffmaster  
          oralib.payrollmaster;  
    by empid;  
run;  
  
libname oralib clear;
```

# SQL Server Example

```
libname sqllib oledb  
    init_string="Provider=SQLOLEDB;  
    password=edu01;  
    Persist Security Info=True;  
    initial catalog=mydata;  
    User ID=dal;  
    data source=edserver"  
    schema=dbo  
    IGNORE_READ_ONLY_COLUMNS=YES;
```

# Analyzing and Reporting on Data

Enhancing Reports with Titles and Footnotes  
Prep Guide Pages 95-100

# Using Titles and Footnotes

*local statements*

TITLE<*n*> "title-text";

FOOTNOTE<*n*> "footnote-text";

```
title1 "Heart Rates for Patients with";
title3 "Increased Stress Tolerance";
footnotel "Data from Treadmill Tests";
footnote3 "1st Quarter Admissions";

proc print data=cert.stress;
  var resthr maxhr rechr;
  where tolerance="I";
run;
```

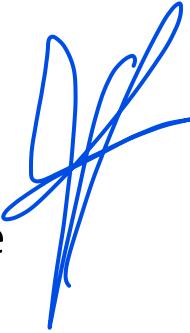
Heart Rates for Patients with  
Increased Stress Tolerance

Obs	RestHR	MaxHR	ReCHR
2	68	171	133
3	78	177	139
8	70	167	122
11	65	181	141
14	74	152	113
15	75	158	108
20	78	189	138

Data from Treadmill Tests

1st Quarter Admissions

# Changing Titles and Footnotes



## TITLE*n* or FOOTNOTE*n*

- replaces value of a previous title or footnote with the same number
- cancels all titles or footnotes with higher(larger) numbers.

# Clearing Titles and Footnotes

```
TITLE;  
FOOTNOTE;
```

clears titles and footnotes

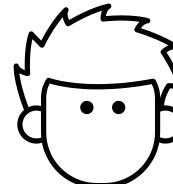
```
ODS NOPROCTITLE;
```

turns off procedure titles

```
title;footnote;  
ods noproctitle;  
proc means data=sashelp.heart;  
    var height weight;  
run;
```



It's a good practice  
to clear all titles  
and footnotes at  
the beginning or end  
of a program.





# Creating Titles and Footnotes

This demonstration the use of titles and footnotes.

# Differing Behavior

Did the PROC MEANS have titles?

- Yes (PC SAS)
- No (SAS Studio)

Some procedures automatically add a procedure title.

SAS Studio

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
RestHR	7	72.5714286	5.0284903	65.0000000	78.0000000
MaxHR	7	170.7142857	12.9449383	152.0000000	189.0000000

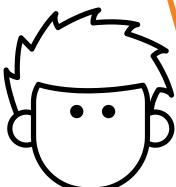
Heart Rates for Patients with

PC SAS

Increased Stress Tolerance

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
RestHR	7	72.5714286	5.0284903	65.0000000	78.0000000
MaxHR	7	170.7142857	12.9449383	152.0000000	189.0000000



It depends. SAS Studio automatically clears existing titles before a new program is submitted. PC SAS does not.

# Lesson Quiz



1. In this PROC CONTENTS output, what is the default length of the **Birth\_Date** column?
  - a. 4 bytes
  - b. 8 bytes
  - c. 32,767 bytes
  - d. It does not have a default length.

#	Variable	Type
4	Birth_Date	Num
3	Customer_Address	Char
1	Customer_ID	Num
2	Customer_Name	Char

1. In this PROC CONTENTS output, what is the default length of the **Birth\_Date** column?
- a. 4 bytes
  - b. 8 bytes
  - c. 32,767 bytes
  - d. It does not have a default length.

#	Variable	Type
4	Birth_Date	Num
3	Customer_Address	Char
1	Customer_ID	Num
2	Customer_Name	Char

2. Which LIBNAME statement has the correct syntax?
- a. `libname reports "filepath/workshop";`
  - b. `libname orion filepath/workshop;`
  - c. `libname 3456a "filepath/workshop";`

2. Which LIBNAME statement has the correct syntax?

- a. `libname reports "filepath/workshop";`
- b. `libname orion filepath/workshop;`
- c. `libname 3456a "filepath/workshop";`

3. Which of the following tables is available at the beginning of a new SAS session?
  - a. sales
  - b. work.newsalesemps
  - c. sashelp.class

3. Which of the following tables is available at the beginning of a new SAS session?
- a. sales
  - b. work.newsalesemps
  - c. sashelp.class

4. In this table, what type of column is Employee\_ID?

- a. character
- b. numeric
- c. temporary
- d. missing

Obs	Employee_ID	Last	Salary
1	.	Ralston	29250
2	120101	Lu	163040
3	120104	Billington	46230
4	120105	Povey	27110
5	120106	Hornsey	.

4. In this table, what type of column is Employee\_ID?

- a. character
- b. numeric
- c. temporary
- d. missing

Obs	Employee_ID	Last	Salary
1	.	Ralston	29250
2	120101	Lu	163040
3	120104	Billington	46230
4	120105	Povey	27110
5	120106	Hornsey	.

5. Which statement about SAS dates is false?
- a. A SAS date is one of three of SAS column types: numeric, character, and date.
  - b. SAS dates represent the number of days from January 1, 1960.
  - c. SAS date values can be positive or negative.
  - d. SAS date values can be used in calculations.

5. Which statement about SAS dates is false?

- a. A SAS date is one of three of SAS column types: numeric, character, and date.
- b. SAS dates represent the number of days from January 1, 1960.
- c. SAS date values can be positive or negative.
- d. SAS date values can be used in calculations.

7. Which library name (libref) is valid?

- a. 2010Car
- b. car/2010
- c. car2010
- d. cars\_2010

7. Which library name (libref) is valid?

- a. 2010Car
- b. car/2010
- c. car2010
- d. cars\_2010

8. To disassociate a libref that you previously assigned, you can use the UNASSIGN option in the LIBNAME statement.
  - a. True
  - b. False

8. To disassociate a libref that you previously assigned, you can use the UNASSIGN option in the LIBNAME statement.
  - a. True
  - b. False

10. In which portion of a SAS table are the following found?

- name of the table
- type of the column **Salary**
- creation date of the table

- a. descriptor portion
- b. data portion

10. In which portion of a SAS table are the following found?

- name of the table
- type of the column **Salary**
- creation date of the table

a. descriptor portion

b. data portion

# SAS Lesson 03

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# Changing and Canceling Titles and Footnotes

PROC PRINT Code

Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	<p>The First Line The Second Line</p>
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	<p>The First Line The Next Line</p>
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

PROC PRINT Code

Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	The First Line The Next Line
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	The First Line The Next Line
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	The Top Line
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	The First Line The Next Line
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	The Top Line
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	The First Line The Second Line
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	The First Line The Next Line
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	The Top Line
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	The Top Line  The Third Line

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	<p>The First Line The Second Line</p>
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	<p>The First Line The Next Line</p>
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	<p>The Top Line</p>
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	<p>The Top Line The Third Line</p>

# Changing and Canceling Titles and Footnotes

## PROC PRINT Code

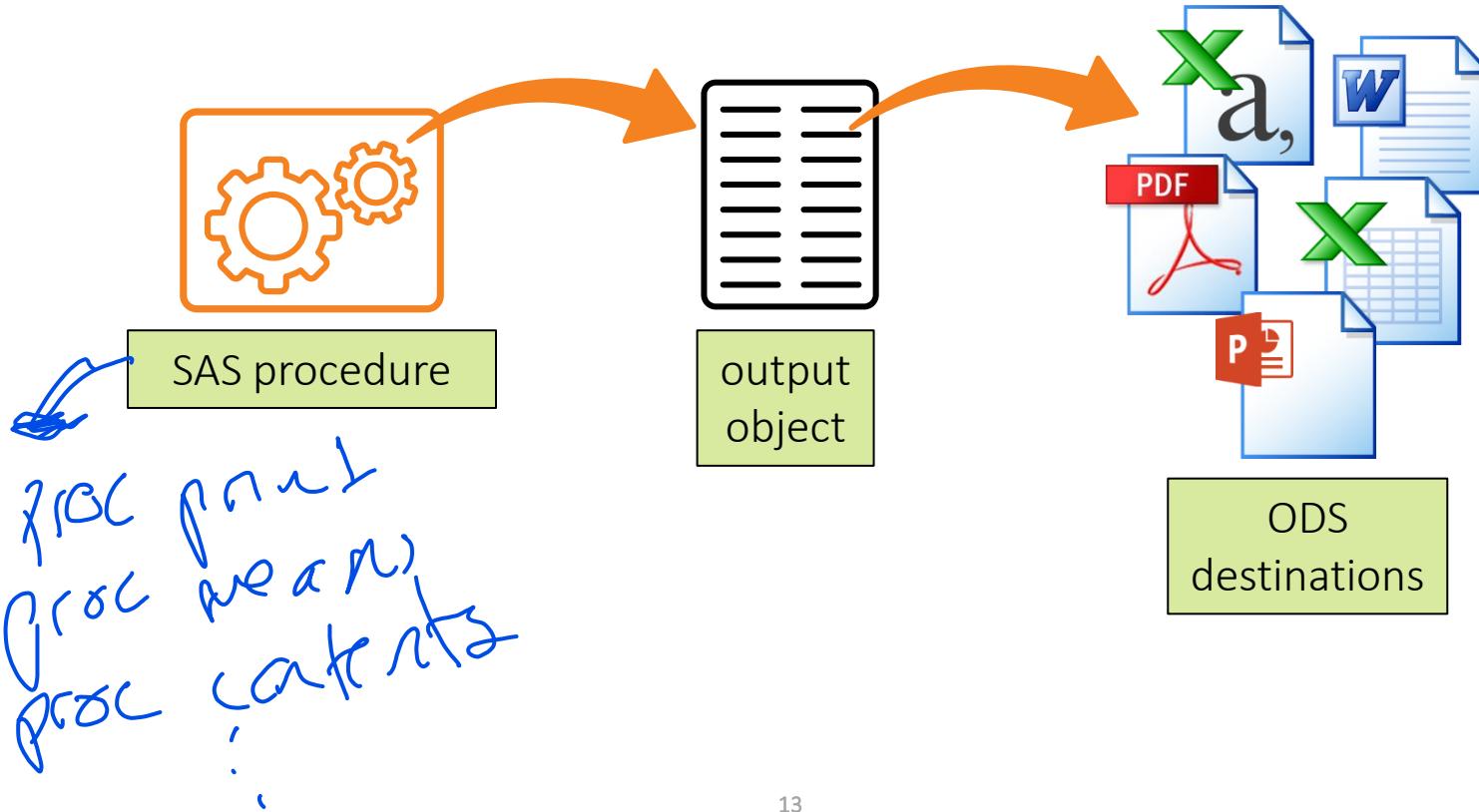
## Resultant Title(s)

<pre>proc print data=orion.sales;   title1 'The First Line';   title2 'The Second Line'; run;</pre>	<p>The First Line The Second Line</p>
<pre>proc print data=orion.sales;   title2 'The Next Line'; run;</pre>	<p>The First Line The Next Line</p>
<pre>proc print data=orion.sales;   title 'The Top Line'; run;</pre>	<p>The Top Line</p>
<pre>proc print data=orion.sales;   title3 'The Third Line'; run;</pre>	<p>The Top Line The Third Line</p>

# Creating Output

Prep Guide Chapter 16

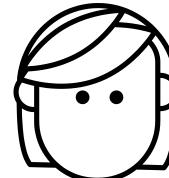
# Using the SAS Output Delivery System



# Using the SAS Output Delivery System



You can create  
different file types  
by changing  
the destination  
in the ODS statement.



# Default ODS Destination

The LISTING destination **was** the default ODS destination until late 9.2. Since then it has been **HTML**.

The SAS System  
21:35 Wednesday, September 30, 2020  
The MEANS Procedure  
Analysis Variable : Age

N	Mean	Std Dev	Minimum	Maximum
19	13.3157895	1.4926722	11.0000000	16.0000000

The SAS System  
The MEANS Procedure  
Analysis Variable : Age

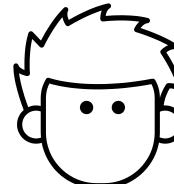
N	Mean	Std Dev	Minimum	Maximum
19	13.3157895	1.4926722	11.0000000	16.0000000

# Multiple Destinations (Club Sandwich?)

Output can be sent to many destinations.

```
ods html close;  
ods pdf file='example.pdf';  
ods rtf file='example.rtf';  
  
proc freq data=orion.sales;  
    tables Country;  
run;  
  
ods pdf close;  
ods rtf close;
```

BEWARE of sending large output to the default HTML destination and another destination.



To view the results, all destinations must be closed.

# Multiple Destinations

Use `_ALL_` in the ODS CLOSE statement to close all open destinations including the DEFAULT destination.

```
ods listing;
ods pdf file='example.pdf';
ods rtf file='example.rtf';

proc freq data=orion.sales;
    tables Country;
run;

ods _all_ close;
ods html path="%qsysfunc(pathname(work))";
```

# Default ODS Destination

A warning will appear in the SAS log if the default destination is closed and no other destinations are active.

## Partial SAS Log

```
23 ods _ALL_ close;  
24  
25 proc freq data=orion.sales;  
26   tables Country;  
27 run;
```

WARNING: No output destinations active.

NOTE: There were 165 observations read from the data set ORION.SALES.

# Multiple Procedures



Output from many procedures can be sent to multiple ODS destinations, even of the same type. Use IDs to differentiate.

```
ods pdf (ID=both) file='both.pdf';
ods pdf (ID=one) file='one.pdf';

proc freq data=sashelp.class;
    tables sex;
run;

ods pdf (one) close;

proc means data=sashelp.class;
    var age;
run;

ods pdf (both) close;
```

# STYLE= Option

Use a STYLE= option in the ODS destination statement to specify a style definition.

```
ODS destination FILE = 'filename.ext'  
      STYLE = style-definition;
```

- A *style definition* describes how to display the presentation aspects such as colors and fonts of SAS output.
- STYLE= cannot be used with the LISTING destination.

# SAS Supplied Style Definitions

```
proc print data=sashelp.vstyle;  
run;
```

Use the SASHELP  
library to find styles  
available on your  
system.



# Exporting Results to PDF

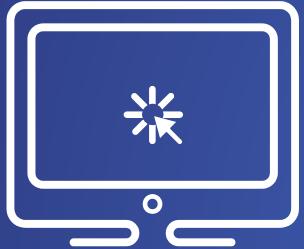
```
ODS PDF FILE="filename.pdf"  
    STARTPAGE=NO  
    CONTENTS=YES  
    BOOKMARKLIST=HIDE  
    PDFTOC=n;  
ODS PROCLABEL "label";  
/* SAS code that produces output */  
ODS PDF CLOSE;
```



The PDF destination is ideal for reporting because the layout can be precisely controlled.

## ~~My dev's have never seen a new page~~ Exporting Results to PDF ~~~ & remove ods pdf w/ option start by = NOW~~

- STARTPAGE=YES|NO|NOW controls when new pages are created
- CONTENTS=YES specifies that a printable table of contents is created
- BOOKMARKLIST=HIDE|NONE|SHOW controls bookmark list within file
- PDFTOC=*n* controls level of bookmarks that are open
- NOTOC turns off both CONTENTS and BOOKMARKLIST
- ODS PROCLABEL "*label*"; defines label for bookmark
- ODS PDF statement can be used multiple times while open to change options.



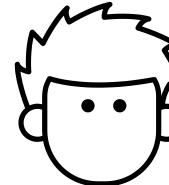
# Exporting Results to PDF

This demonstration illustrates using the ODS PDF destination to export reports to a PDF file.

# Exporting Results to HTML

```
ODS HTML PATH= "c:\user\certuser"  
    BODY|FILE="filename.html"  
    (URL=none)  
    CONTENTS="toc.html"  
    FRAME="frame.html"  
;  
/* SAS code that produces output */  
ODS HTML CLOSE;
```

The HTML destination creates web pages for viewing in a web browser.



# Exporting Results to HTML

- **PATH=** specifies the location of the files
- **BODY|FILE=** specifies the file that contains the html output
- **CONTENTS=** name of table of contents file with links to output
- **FRAME=** name of file that integrates body and table of contents
- **URL=** sub-option to control location of body and contents files (NOTE: Use relative URLs to allow HTML package to be easily relocated.)



# Exporting Results to HTML

This demonstration illustrates using the ODS HTML destination to export reports to a Web Page.

# 2019 SAS Global Forum

## Exporting Results to Excel

```
ODS EXCEL FILE="filename.xlsx" STYLE=style
  OPTIONS(SHEET_NAME='label'
          EMBEDDED_FOOTNOTES='on'
          EMBEDDED_TITLES='on'
          SHEET_INTERVAL='bygroup'
          SUPPRESS_BYLINES='yes');

/* SAS code that produces output */

ODS EXCEL CLOSE;
```

By default, the results from each procedure are on separate worksheets in the Excel file.



# Exporting Results to Excel (Sub-Options)

- **SHEET\_NAME=** specifies the full name of the **next** worksheet
- **SHEET\_LABEL=** specifies the **prefix** for the worksheet names
  - Especially useful with **bygroup** interval
- **EMBEDDED\_FOOTNOTES=** specifies whether footnotes appear in worksheet
- **EMBEDDED\_TITLES=** specifies whether titles appear in worksheet
- **SHEET\_INTERVAL=** specifies the criteria for when a new worksheet is created *(Do never put anything on sheet)*
- **SUPPRESS\_BYLINES=** specifies whether BY lines appear in worksheet



# Exporting Results to Excel

This demonstration illustrates using the ODS EXCEL destination to export reports to multiple worksheets in an Excel workbook.

# Exporting Output to PowerPoint and Microsoft Word

```
ODS POWERPOINT FILE="filename.pptx" STYLE=style;  
/* SAS code that produces output */  
ODS POWERPOINT CLOSE;
```

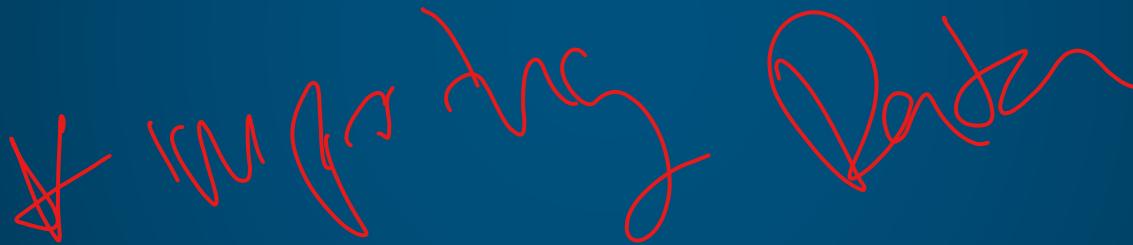
```
ODS RTF FILE="filename.rtf" STARTPAGE=NO;  
/* SAS code that produces output */  
ODS RTF CLOSE;
```

RTF files can be read by word processing software such as Microsoft Word.



# Accessing Data

Importing Data into SAS – Prep Guide Chapter 4



# Assigning a File Reference

You can use the *FILENAME statement* to assign a file reference name (fileref) to an external file.

General form of the FILENAME statement:

```
FILENAME fileref 'external-file' <options>;
```

Rules for naming a fileref:

- The name must be 8 characters or less.
- The name must begin with a letter or underscore.
- The remaining characters must be letters, numerals, or underscores.

# Assigning and Using a Fileref

Windows Example:

## Assigning

```
filename pdfrep1 'c:\users\certuser\Assign6.pdf' ;
```

```
FILENAME OUT FTP '/home/ftpas/dialog.txt'
      host='ftpsrv.tamu.edu'
      user='ftpas' pass='ftppass' lrecl=2437
      rcmd='site umask 022'
      /* Set permissions to -rw-r--r--      */
;
```

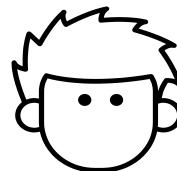
# Assigning and Using a Fileref

Windows Example:

Using

```
ods pdf file = pdfrep1 notoc;
```

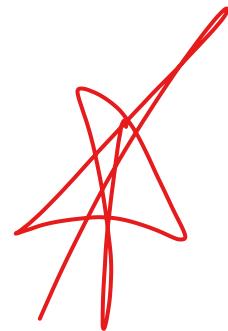
Assigning filerefs at  
the top of the  
program keeps  
paths in one place!



Do not put the  
fileref in quotes like  
you would the full  
path.



# Reviewing Concepts



- Libref (libname) = alias to a collection of tables
- Fileref (filename) = alias to a single file
- Raw text files are not considered tables and cannot be accessed through a libref
- Both librefs and filerefs can be read from and written to
- Same naming rules apply to both

# Lesson Quiz



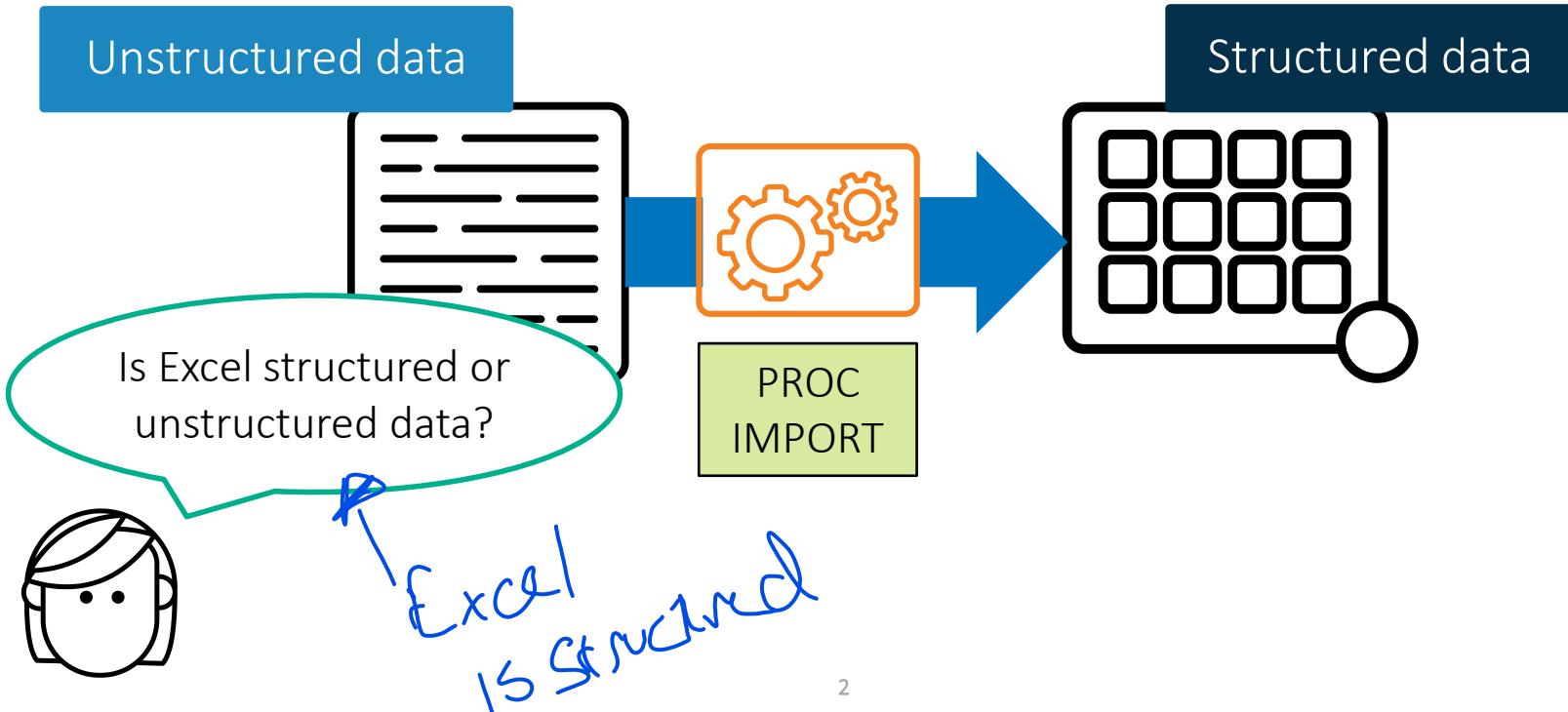
6. Which LIBNAME statement has the correct syntax for reading a Microsoft Excel file?
- a. `libname excel "filepath/myexcelfile";`
  - b. `libname mydata xlsx "filepath/myexcelfile";`
  - c. `libname mydata xlsx "filepath/field_data.xlsx";`

6. Which LIBNAME statement has the correct syntax for reading a Microsoft Excel file?
- a. `libname excel "filepath/myexcelfile";`
  - b. `libname mydata xlsx "filepath/myexcelfile";`
  - c. `libname mydata xlsx "filepath/field_data.xlsx";`

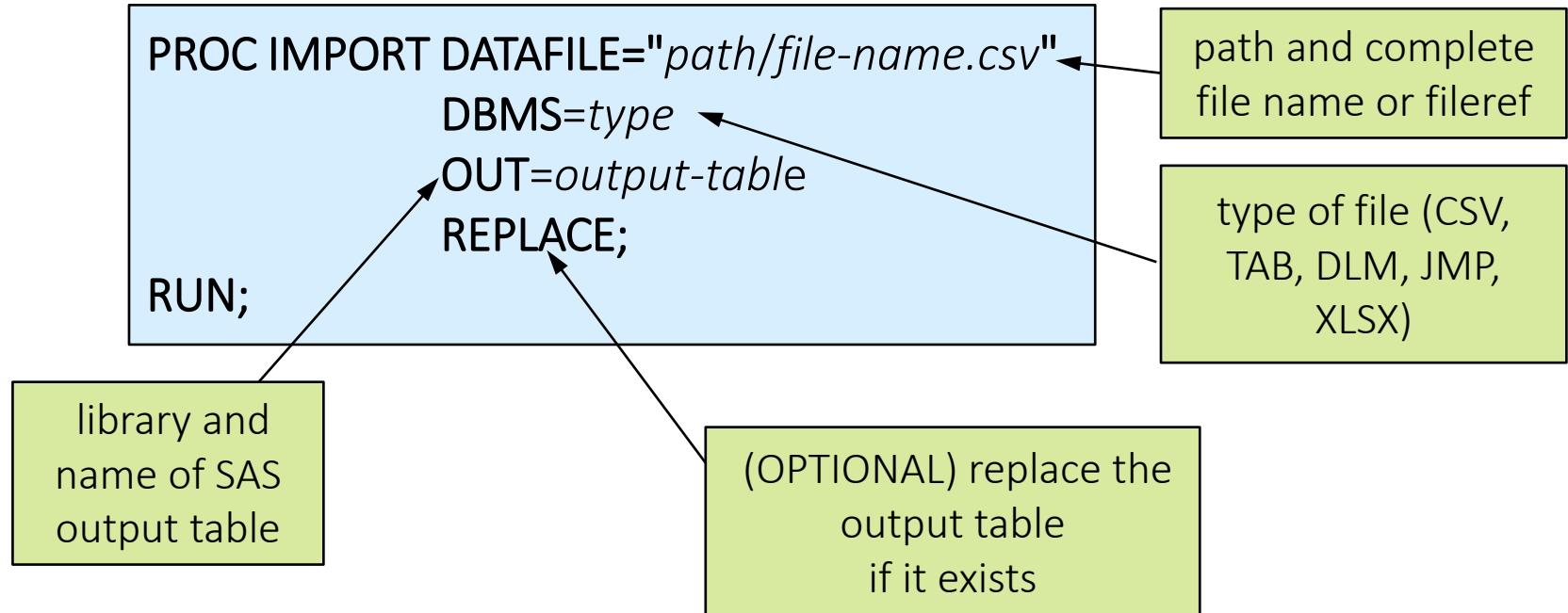
# SAS Lesson 04

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# Importing Unstructured Data



# Importing Unstructured Data – General Form



# Importing Unstructured Data – Optional Statements

specifies the number of rows used to determine column type and length (default = 20)

controls extraction of variable names from first row

which row SAS begins to read data

```
PROC IMPORT DATAFILE="path/file-name.csv"  
    DBMS=type  
    OUT=output-table;  
  
    GUESSINGROWS=n | MAX;  
    GETNAMES=NO;  
    DATAROW=n;  
    DELIMITER=" ";  
  
RUN;
```

delimiter for DLM or TAB ('09'x ASCII or '05'x EBCDIC)

# Importing a Comma-Delimited (CSV) File

```
1 "Africa","Boot","Addis Ababa","12","$29,761","$191,821","$769".....CRLF
2 "Asia","Boot","Bangkok","1","$1,996","$9,576","$80".....CRLF
3 "Canada","Boot","Calgary","8","$17,720","$63,280","$472".....CRLF
4 "Central America/Caribbean","Boot","Kingston","33","$102,372","$393,376","$4,454".....
5 "Eastern Europe","Boot","Budapest","22","$74,102","$317,515","$3,341".....
6 "Middle East","Boot","Al-Khobar","10","$15,062","$44,658","$765".....
7 "Pacific","Boot","Auckland","12","$20,141","$97,919","$962".....
8 "South America","Boot","Bogota","19","$15,312","$35,805","$1,229".....
9 "United States","Boot","Chicago","16","$82,483","$305,061","$3,735".....
10 "Western Europe","Boot","Copenhagen","2","$1,663","$4,657","$129".....
11
```

# Importing a Comma-Delimited (CSV) File

limit  
observations  
for testing

```
OPTIONS OBS=5;  
FILENAME CSVIN="c:\users\student1\cert\boot.csv";  
PROC IMPORT DATAFILE=CSVIN  
    DBMS=CSV  
    OUT=shoes  
    REPLACE;  
    GETNAMES=no;  
RUN;  
OPTIONS OBS=max;
```

Reset option



# Importing a Tab-Delimited File

This demonstration illustrates importing a tab-delimited file and creating a new SAS table using PROC IMPORT.

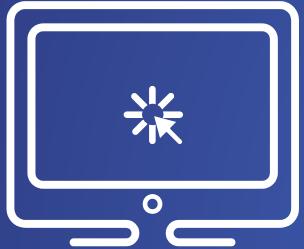
# Importing an Excel File

name of sheet  
that you want  
to import

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

type of file

```
filename xlin "c:\users\student1\cert\boots.xlsx" ;  
proc import datafile= xlin  
            dbms=xlsx  
            out=work.bootsales replace;  
            sheet=boot;  
run;
```



# Importing an Excel File

This demonstration illustrates importing a file from Excel and creating a new SAS table using PROC IMPORT.



## Discussion

What is the difference between using the XLSX LIBNAME engine and PROC IMPORT to read Excel data in a SAS program?

# SAS LIBNAME Engines for Microsoft Excel

Comparison of the SAS LIBNAME Engines That Access Microsoft Excel Data

Feature	EXCEL LIBNAME Engine	PCFILES LIBNAME Engine	XLSX LIBNAME Engine
Host support	Microsoft Windows	Windows, UNIX	Windows, UNIX, Studio
Requires the SAS PC Files Server	No	Yes	No
Requires Microsoft Access Database Engine (ACE)	Yes *	Yes	No
Supports SAS LIBNAME options	Yes	Yes	Limited
Supports SAS data set options	Yes	Yes	Limited
Supports SAS SQL procedure and pass-through	Yes	Yes	No
Reads data support for file types	.xlsx, .xlsm, .xls	.xlsx, .xlsm, .xls	.xlsx
Creates data support for file types (new table)	.xlsx, .xlsm, .xls	.xlsx, .xlsm, .xls	.xlsx
Updates data support for file types	.xlsx, .xlsm, .xls	.xlsx, .xlsm, .xls	.xlsx

\* Requires bit consistency

# Using a Library to Read Excel Files (Review)

```
LIBNAME libref XLSX "path/file-name.xlsx";
```

name of  
the library

The XLSX engine  
reads Excel files.

The path must include  
the complete file  
name and extension.

```
libname xlclass xlsx "s:/workshop/data/class.xlsx";
```

The XLSX engine requires a license  
for SAS/ACCESS Interface to PC Files.

# Using the EXCEL Engine to Read Excel Files (PC SAS)

```
LIBNAME libref EXCEL "path/file-name.xlsx";
```

name of  
the library

The EXCEL engine  
is the default to  
read Excel files.

The path must include  
the complete file  
name and extension.

```
libname xlclass excel "s:/workshop/data/class.xlsx";
```

The EXCEL engine requires a license for SAS/ACCESS  
Interface to PC Files and bit agreement.

# Using PCFILES Server to Read Excel Files (PC SAS)

```
LIBNAME libref PCFILES type=excel path="path/file-name.xlsx";
```

name of  
the library

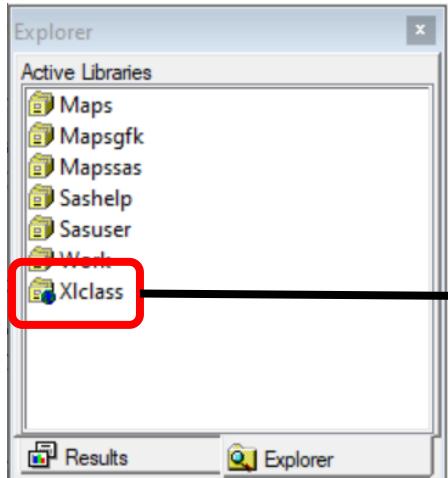
The PCFILES engine  
requires additional  
parameters.

The path must include  
the complete file  
name and extension.

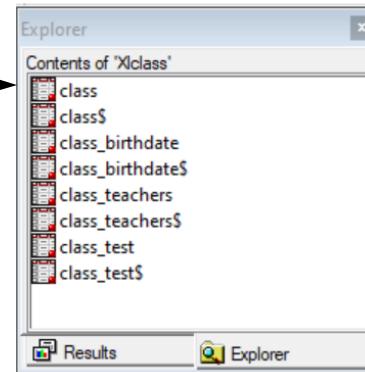
```
libname xlclass pcfiles type=excel  
path="s:/workshop/data/class.xlsx";
```

Use PCFILES to access Excel workbooks when the ACE and OS have a bit mismatch. Requires installation of PCFILES server service on Windows.

# SAS Explorer Window



Each worksheet in the Excel workbook is treated as though it is a SAS data set.



Worksheet names appear with a dollar sign at the end of the name with EXCEL and PCFILES engines.

# The CONTENTS Procedure

```
proc contents data=xlclass._all_ nods;  
run;
```

The CONTENTS Procedure			
Directory			
Libref	XLCLASS		
Engine	EXCEL		
Physical Name	C:\Users\kinchel\Documents\PC SAS\Prog1\Data\data\class.xlsx		
User	Admin		
#	Name	Member Type	DBMS Member Type
1	class	DATA	TABLE
2	class\$	DATA	TABLE
3	class_birthdate	DATA	TABLE
4	class_birthdate\$	DATA	TABLE
5	class_teachers	DATA	TABLE
6	class_teachers\$	DATA	TABLE
7	class_test	DATA	TABLE
8	class_test\$	DATA	TABLE

# SAS Name Literals

By default, special characters such as the \$ are not allowed in data set names.

SAS name literals enable special characters to be included in data set names.

A *SAS name literal* is a name token that is expressed as a string within quotation marks, followed by the letter n.

```
proc contents data=xlclass.'class_birthdate$n'
```

SAS name literal

# The CONTENTS Procedure

```
proc contents data=xlclass.'class_birthdate$n';
run;
```

The CONTENTS Procedure

Data Set Name	XLCLASS.'class_birthdate\$n'	Observations	.
Member Type	DATA	Variables	6
Engine	EXCEL	Indexes	0
Created	.	Observation Length	0
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

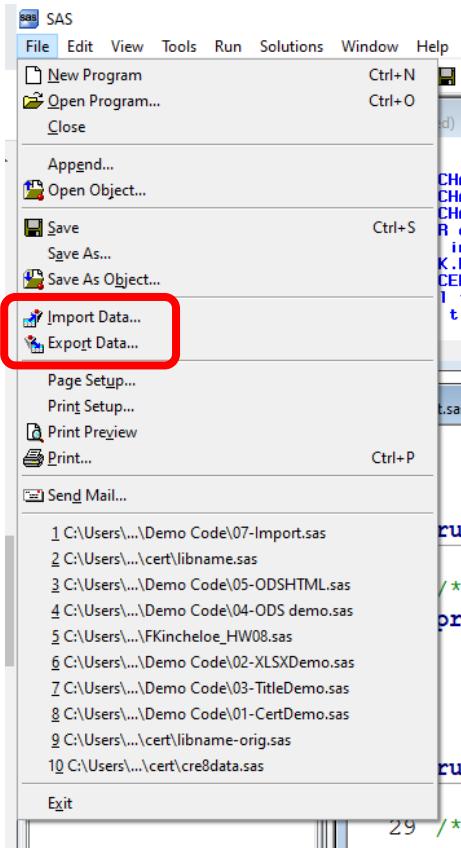
Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
3	Age	Num	8			Age
6	Birthdate	Num	8	DATE9.	DATE9.	Birthdate

# Exporting Results

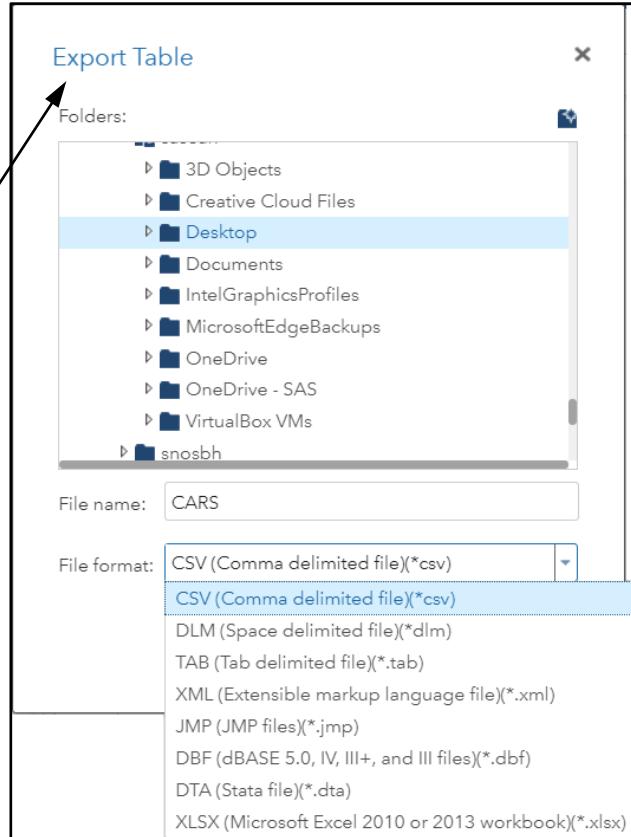
Exporting Data – Prep Guide page 356

# Exporting Data Using Point-and-Click Tools



Right-click a table in SAS Studio and select Export.

Import wizard is under Tasks and Utilities



# Exporting Data Using Code

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

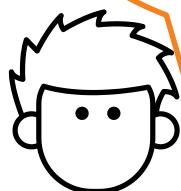
data source name (SAS table)

Output fileref or path and name

tells SAS how to format the output (CSV, TAB, DLM, XLSX)

overwrite output if it exists

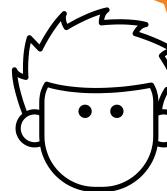
Column names are automatically written as the first row of the output file.



# Exporting Data Using Code

```
proc export data=sashelp.cars  
  outfile= "/folders/myfolders/output/cars.txt"  
  dbms=tab replace;  
run;
```

Remember that  
the path is relative  
to the location  
of SAS.

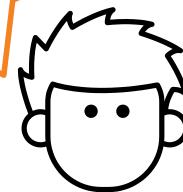


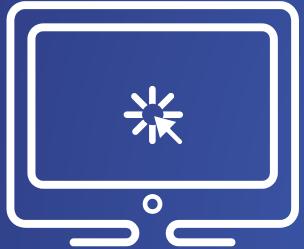
# Exporting Data with a LIBNAME Engine

```
libname myxl xlsx "/folders/myfolders/output/cars.xlsx";  
  
data myxl.asiacars;  
  set sashelp.cars;  
  where origin='Asia';  
run;  
  
libname myxl clear;
```

This code extracts data and writes it to the **cars** workbook on a tab named **asiacars**.

defines a library to the Microsoft Excel workbook that you are creating





# Exporting Data to an Excel Workbook

This demonstration illustrates using the XLSX LIBNAME engine and PROC Export to export SAS tables to multiple worksheets in an Excel workbook.



## Discussion

What is the difference between using ODS Excel and PROC EXPORT to create Excel data in a SAS program?

# Accessing Data

Creating data sets from SAS tables

# Using a SAS Data Set as Input

```
data men50.males;  
  set cert.admit;  
  where sex= 'M' and  
        age > 50;  
run;
```

```
DATA output-SAS-data-set;  
  SET input-SAS-data-set;  
  WHERE WHERE-expression;  
RUN;
```

# DATA Statement

The *DATA statement* begins a DATA step and provides the name of the SAS data set to create.

```
data men50.males;
  set cert.admit;
  where sex= 'M' and
        age > 50;
run;
```

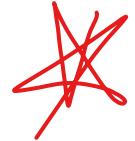
DATA *output-SAS-data-set*;

A DATA step can create temporary or permanent data sets.



- The rules for SAS variable names also apply to data set names.

# SET Statement



The *SET statement* reads observations from an existing SAS data set for further processing in the DATA step.

```
data men50.males;
  set cert.admit;
  where sex= 'M' and
        age > 50;
run;
```

SET *input-SAS-data-set*;

- The SET statement reads all observations and all variables from the input data set.
- Observations are read sequentially, one at a time.
- The SET statement can read temporary or permanent data sets.

# WHERE Statement

The *WHERE statement* selects observations from a SAS data set that meet a particular condition.

```
data men50.males;  
  set cert.admit;  
  where sex= 'M' and  
        age > 50;  
run;
```

WHERE WHERE-expression;

The variables named in the WHERE expression must exist in the input SAS data set.

# Multiple Choice Poll

Considering this DATA step, which statement is true?

```
data us;  
  set orion.sales;  
  where Country='US';  
run;
```

- a. It reads a temporary data set and creates a permanent data set.
- b. It reads a permanent data set and creates a temporary data set.
- c. It contains a syntax error and does not execute.
- d. It does not execute because you cannot work with permanent and temporary data sets in the same step.

# Multiple Choice Poll – Correct Answer

Considering this DATA step, which statement is true?

- a. It reads a temporary data set and creates a permanent data set.
- b.** It reads a permanent data set and creates a temporary data set.
- c. It contains a syntax error and does not execute.
- d. It does not execute because you cannot work with permanent and temporary data sets in the same step.

```
data us;                      /* Create a temporary data set */
  set orion.sales; /* Read a permanent data set */
  where Country='US';
run;
```

# Lesson Quiz



9. What does this code do?

```
proc import datafile="d:/collect817/bird_count.csv"  
          dbms=csv out=bird817 replace;  
run;
```

- a. It creates a SAS table named **Bird817** in the **Work** library from the CSV file **bird\_count** and replaces **Bird817** whenever the CSV file is updated.
- b. It creates a SAS table named **Bird817** in the **Work** library from the CSV file **bird\_count**.
- c. It uses the CSV engine to directly read the data file **bird\_count.csv**.

9. What does this code do?

```
proc import datafile="d:/collect817/bird_count.csv"  
          dbms=csv out=bird817 replace;  
run;
```

- a. It creates a SAS table named **Bird817** in the **Work** library from the CSV file **bird\_count** and replaces **Bird817** whenever the CSV file is updated.
- b. It creates a SAS table named **Bird817** in the **Work** library from the CSV file **bird\_count**.
- c. It uses the CSV engine to directly read the data file **bird\_count.csv**.

4. Which statement disassociates the **sales** libref?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';
```

- a. **libname sales end;**
- b. **libname sales clear;**
- c. **libname sales close;**
- d. **libname sales disassociate;**

4. Which statement disassociates the **sales** libref?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';
```

- a. `libname sales end;`
- b. `libname sales clear;`
- c. `libname sales close;`
- d. `libname sales disassociate;`

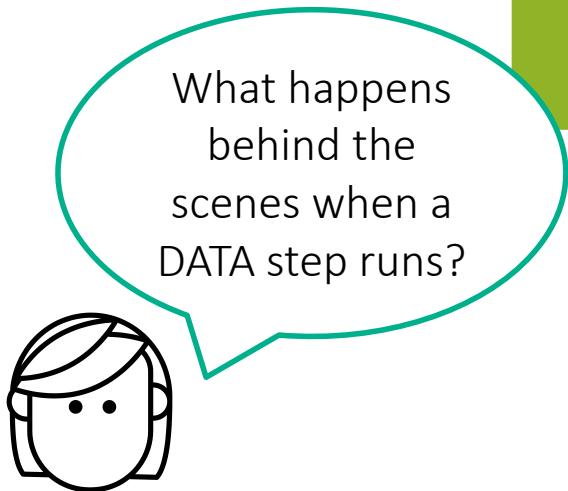
# SAS Lesson 05

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# DATA Step Processing

Understanding DATA Step Processing – Chapter 7

# DATA Step Processing



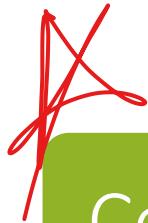
## Compilation

establish data attributes and rules for execution



## Execution

read, manipulate, and write data



# DATA Step Processing: Compilation

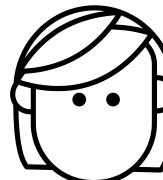
## Compilation

- 1) Check for syntax errors.
- 2) Create the *program data vector (PDV)*, which includes all columns and attributes.
- 3) Establish the specifications for processing data in the PDV during execution.
- 4) Create the descriptor portion of the output table.

PDV

Season N 8	Name \$ 25	StartDate N 8	Ocean \$ 8

The PDV is the magic behind the DATA step's processing power!



# DATA Step Processing: Compilation

```
data storm_complete;
  set pg2.storm_summary_small;
  length Ocean $ 8;
  drop EndDate;
  where Name is not missing;
  Basin=upcase(Basin);
  StormLength=EndDate-StartDate;
  if substr(Basin,2,1)="I" then Ocean="Indian";
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";
  else Ocean="Pacific";
run;
```

Define the library and a name for the output table.

# DATA Step Processing: Compilation

```
data storm_complete;  
  set pg2.storm_summary_small;  
  length Ocean $ 8;  
  drop EndDate;  
  where Name is not missing;  
  Basin=uppercase(Basin);  
  StormLength=EndDate-StartDate;  
  if substr(Basin,2,1)="I" then Ocean="Indian";  
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";  
  else Ocean="Pacific";  
run;
```

Columns are added to the PDV  
in the order in which they  
appear in the input table.

## PDV

Name \$ 15	Basin \$ 2	MaxWind N 8	StartDate N 8	EndDate N 8

Attributes are inherited  
from the input table.

# DATA Step Processing: Compilation

```
data storm_complete;
  set pg2.storm_summary_small;
  length Ocean $ 8;
  drop EndDate;
  where Name is not missing;
  Basin=uppercase(Basin);
  StormLength=EndDate-StartDate;
  if substr(Basin,2,1)="I" then Ocean="Indian";
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";
  else Ocean="Pacific";
run;
```

The remaining columns are added to the PDV in the order in which they appear in the DATA step.

PDV

Each column must have at least a name, type, and length.

Name \$ 15	Basin \$ 2	MaxWind N 8	StartDate N 8	EndDate N 8	Ocean \$ 8	StormLength N 8

# DATA Step Processing: Compilation

```
data storm_complete;
  set pg2.storm_summary_small;
  length Ocean $ 8;
  drop EndDate;
  where Name is not missing;
  Basin=uppercase(Basin);
  StormLength=EndDate-StartDate;
  if substr(Basin,2,1)="I" then Ocean="Indian";
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";
  else Ocean="Pacific";
run;
```

DROP or KEEP statements flag columns that will be excluded from the output table.

## PDV

Name \$ 15	Basin \$ 2	MaxWind N 8	StartDate N 8	EndDate N 8	Ocean \$ 8	StormLength N 8
				D		

# DATA Step Processing: Compilation

```
data storm_complete;
  set pg2.storm_summary_small;
  length Ocean $ 8;
  drop EndDate;
  where Name is not missing;
  Basin=uppercase(Basin);
  StormLength=EndDate-StartDate;
  if substr(Basin,2,1)="I" then Ocean="Indian";
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";
  else Ocean="Pacific";
run;
```

The WHERE statement establishes conditions for which rows will be read from the input table into the PDV.

## PDV

Name	Basin	MaxWind	StartDate	EndDate	Ocean	StormLength
 \$ 15	\$ 2	N 8	N 8	 N 8	\$ 8	N 8

# DATA Step Processing: Compilation

PDV

Name \$ 15	...	StormLength N 8	-N-	-ERROR-
			D	D

Index counter for the DATA step loop (number of iterations).

Changes from 0 to 1 to indicate a data error during execution.

These variables are not written to the output data set.

# DATA Step Processing: Compilation

```
data storm_complete;
  set pg2.storm_summary_small;
  length Ocean $ 8;
  drop EndDate;
  where Name is not missing;
  Basin=upcase(Basin);
  StormLength=EndDate-StartDate;
  if substr(Basin,2,1)="I" then Ocean="Indian";
  else if substr(Basin,2,1)="A" then Ocean="Atlantic";
  else Ocean="Pacific";
run;
```

The descriptor portion is created for the output table.

**work.storm\_complete**

Name	Basin	MaxWind	StartDate	Ocean	StormLength
\$ 15	\$ 2	N 8	N 8	\$ 8	N 8

# DATA Step Processing: Execution

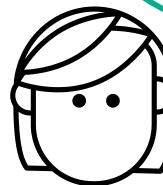
## Execution

- 1) Initialize the PDV.
- 2) Read a row from the input table into the PDV.
- 3) Sequentially process statements and update values in the PDV.
- 4) At the end of the step, write the contents of the PDV to the output table.
- 5) Return to the top of the DATA step.

```
data output-table;  
  set input-table;  
  ...other statements...  
run;
```

Implicit OUTPUT;  
Implicit RETURN;

Automatic  
looping makes  
processing data  
easy!



# DATA Step Processing: Execution

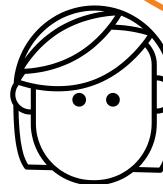
## Iteration 2+

- 1) Re-Initialize the PDV\*.
- 2) Read a row from the input table into the PDV.
- 3) Sequentially process statements and update values in the PDV.
- 4) At the end of the step, write the contents of the PDV to the output table.
- 5) Return to the top of the DATA step.

```
data output-table;  
  set input-table;  
  ...other statements...  
run;
```

Implicit OUTPUT;  
Implicit RETURN;

\*Variables read from data sets are NOT re-initialized!



# DATA Step Processing in Action

The screenshot shows the SAS DATA STEP Source Level Debugger interface. The top bar includes the SAS logo, file menu (File, Edit, View, Tools, Solutions, Window, Help), and a toolbar with various icons. The left pane is the Explorer window showing 'Contents of 'Pg2'' with several datasets listed. The main area has two panes: 'DEBUGGER LOG' at the top and 'PROC DEBUGGER SOURCE' at the bottom. The LOG pane displays the following text:

```
DATA STEP Source Level Debugger  
Stopped at 1 line 9 column 5  
> e _all_  
Name =  
Basin =  
MaxKind = :  
StartDate = .  
EndDate = .  
Ocean =  
StormLength = .  
ERROR_ = 0  
_N_ = 1
```

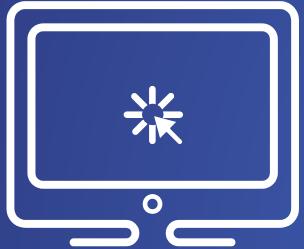
The SOURCE pane shows the following PROC DEBUGGER SOURCE code:

```
8 data storm complete / debug;  
9   set pg2.storm_summary_small;  
10  length Ocean $ 8;  
11  drop EndDate;  
12  where Name is not missing;  
13  Basin=upcase(Basin);  
14  StormLength=EndDate-StartDate;  
15  if substr(Basin,2,1)="I" then Ocean="Indian";  
16  else if substr(Basin,2,1)="A" then Ocean="Atlantic";  
17  else Ocean="Pacific";  
18 run;
```

Annotations on the right side of the interface provide additional context:

- A green box states: "Only available in PC SAS or Enterprise Guide."
- A green box states: "Add / debug to the end of the DATA step.."
- A large green callout bubble states: "You can watch execution happen one statement at a time in the PC SAS DATA step debugger."

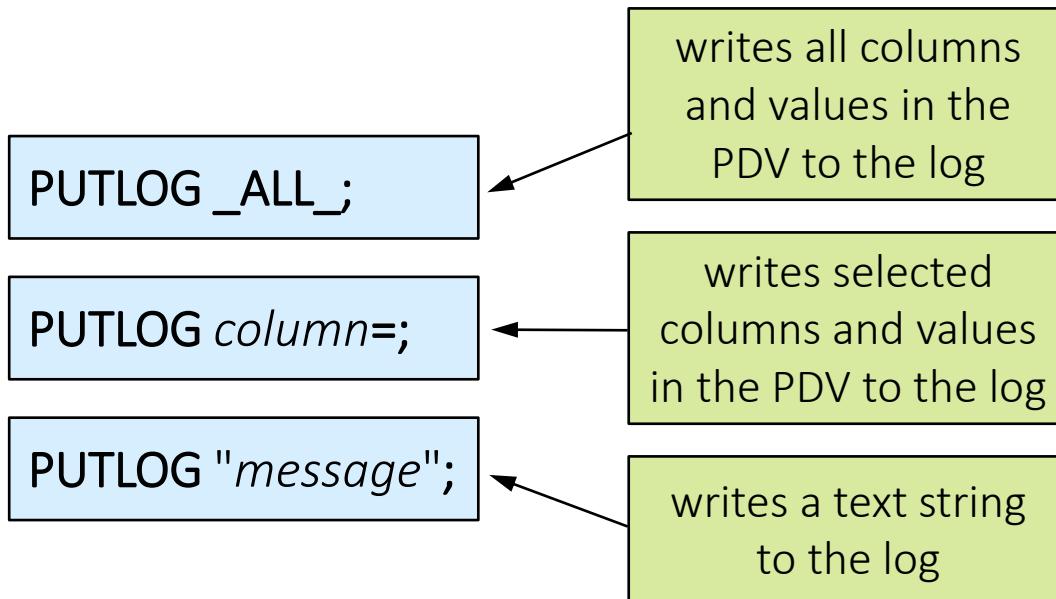
At the bottom right is a small cartoon character wearing a thinking cap.



# DATA Step Processing

This demonstration illustrates using the DATA step debugger in PC SAS to observe the process of execution.

# Viewing Execution in the Log

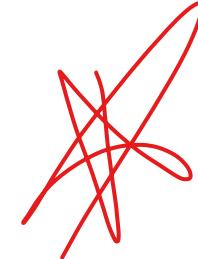


If you don't have the interactive debugger, use the PUTLOG statement to write information about execution to the log.



# PUT vs. PUTLOG

## Prep Guide Chapter 5



### PUT

- Writes to output files if open
- Can be used to control text in output files
- Writes to log only if no file destination is open

### PUTLOG

- Only writes to log

# Viewing Execution in the Log

```
data storm_complete;
  set pg2.storm_summary_small(obs=2);
  putlog "PDV after SET Statement";
  putlog _all_;
  ...

```

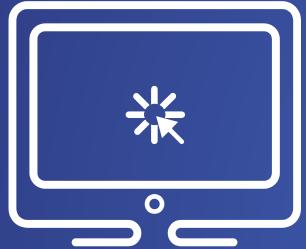
The OBS= data set option limits the observations that are read.

```
PDV after SET Statement
Name=AGATHA Basin=EP MaxWind=115 StartDate=09JUN1980
EndDate=15JUN1980 Ocean=  StormLength=. _ERROR_=0 _N_=1
PDV after SET Statement
Name=ALBINE Basin=SI MaxWind=. StartDate=27NOV1979
EndDate=06DEC1979 Ocean=  StormLength=. _ERROR_=0 _N_=2
```



# DATA Step Processing

This demonstration illustrates using log messages with the DATA step to observe the process of execution.



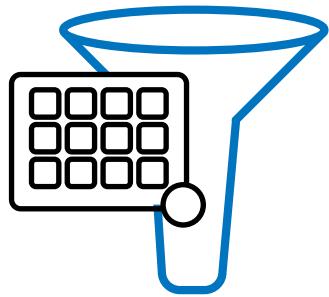
# DATA Step Processing

This demonstration illustrates the behavior caused by unbalanced quotation marks.

# DATA Step Processing

Filtering Rows

# Filtering Rows with the WHERE Statement



```
DATA outset;  
  SET inset;  
  WHERE expression;  
RUN;
```

filters rows in  
the results based  
on the expression

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

If *expression* is true,  
include the row  
in the results.



# Using Basic Operators in an Expression

WHERE *expression*;

column    operator    value

= or EQ

< or LT

^= or ~= or NE

>= or GE

> or GT

<= or LE

Type = "SUV"

Type EQ "SUV"

MSRP <= 30000

MSRP LE 30000

# Specifying Values in an Expression

**WHERE** *expression;*

column

operator

value

Character values are case sensitive and must be enclosed in double or single quotation marks.

Numeric values must be standard numeric (that is, no symbols).

**Type** = "SUV"

**Type** = 'Wagon'

**MSRP** <= 30000

# Specifying Values in an Expression

**WHERE** *expression;*

column    operator    value

Use a *SAS date constant* when you want to evaluate a SAS date value in an expression.



"ddmmmyyyy"d

**where date > "01JAN2015"d;**

**where date > "1jan15"d;**

# Combining Expressions

```
proc print data=sashelp.cars;  
  var Make Model Type MSRP MPG_City MPG_Highway;  
  where Type="SUV" and MSRP <= 30000;  
run;
```

Expressions can be combined with AND or OR.

Obs	Make	Model	Type	MSRP	MPG_City	MPG_Highway
48	Buick	Rendezvous CX	SUV	\$26,545	19	26
67	Chevrolet	Tracker	SUV	\$20,255	19	22
121	Ford	Explorer XLT V6	SUV	\$29,670	15	20
122	Ford	Escape XLS	SUV	\$22,515	18	23
152	Honda	Pilot LX	SUV	\$27,560	17	22

# Using the IN Operator



```
WHERE col-name IN (value-1,...,value-n);  
WHERE col-name NOT IN (value-1,...,value-n);
```

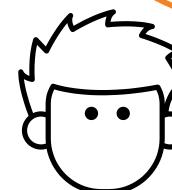
Values can be  
character or  
numeric.

```
where Type="SUV" or Type="Truck" or Type="Wagon";
```

```
where Type in ("SUV", "Truck", "Wagon");
```

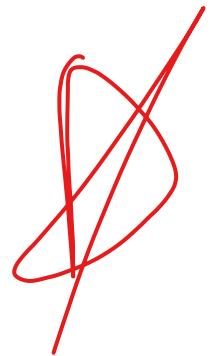
```
where Type in ("SUV" "Truck" "Wagon");
```

All three of these  
statements have  
the same result.



## Special WHERE Operators

*Special WHERE operators* are operators that can only be used in a where-expression\*.



Symbol	Mnemonic	Definition
	BETWEEN-AND	an inclusive range
	IS NULL	missing value
	IS MISSING	missing value
?	CONTAINS	a character string
	LIKE	a character pattern

\* Do not work on IF statements!

# Using Special WHERE Operators

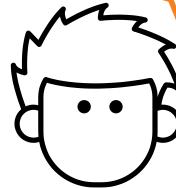
**WHERE** *col-name* **BETWEEN** *value-1* **AND** *value-2*;

```
where age between 20 and 39;
```

```
where 20 <= age <= 39;
```

includes rows with values  
*between and including* the  
endpoints that you specify

For character values,  
the range is based  
on the alphabet.



# Using Special WHERE Operators

```
WHERE col-name IS NULL;
```

```
WHERE col-name IS NOT NULL;
```

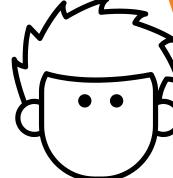
```
WHERE col-name IS MISSING;
```

```
WHERE col-name IS NOT MISSING;
```

```
where age is missing;
```

```
where name is not missing;
```

These operators work  
for both character  
and numeric  
missing values.



# Using Special WHERE Operators



`WHERE col-name CONTAINS "value";`

`where Job_Title contains 'Rep' ;`

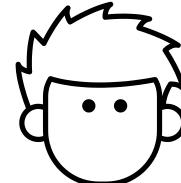
The **position** of the substring within the variable's values is **not important**.

The operator is **case sensitive** when you make comparisons.

Republican Party

House of Representatives

The *CONTAINS (?)* operator selects observations that include the specified substring.



# Using Special WHERE Operators



`WHERE col-name LIKE "value";`

`where City like "New%" ;`

New York
New Delhi
Newport
Newcastle
New

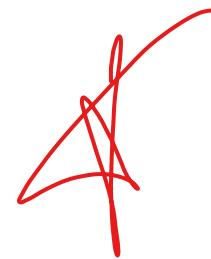
wildcard  
for any  
number of  
characters

`where City like "Sant_%" ;`

Santa Clara
Santa Cruz
Santo Domingo
Santo Tomas

wildcard  
for a single  
character

# Using Special WHERE Operators



How do you search for % and \_ in your data?

Beginning with 9.2, an escape character forces literal search for % and \_.

Example:

```
where Name like 'A/_C' escape '/' ;
```



# Filtering Rows with Basic Operators

This demonstration illustrates using the WHERE statement and basic operators to subset rows in a procedure.

# Lesson Quiz



1. Which statement is false concerning the compilation phase of the DATA step?
  - a. Initial values are assigned to the columns.
  - b. The program data vector (PDV) is created.
  - c. The DATA step is checked for syntax errors.
  - d. The descriptor portion of the output table is created.

1. Which statement is false concerning the compilation phase of the DATA step?
  - a. Initial values are assigned to the columns.
  - b. The program data vector (PDV) is created.
  - c. The DATA step is checked for syntax errors.
  - d. The descriptor portion of the output table is created.

2. Which statement is not a compile-time-only statement?

- a. KEEP
- b. LENGTH
- c. SET
- d. WHERE

2. Which statement is not a compile-time-only statement?

- a. KEEP
- b. LENGTH
- c. SET
- d. WHERE

3. Which statement is true concerning the execution phase of the DATA step?
- a. Data is processed in the program data vector (PDV).
  - b. An implied OUTPUT occurs at the top of the DATA step.
  - c. An implied REINITIALIZE occurs at the bottom of the DATA step.
  - d. Columns read from the input table are set to missing when SAS returns to the top of the DATA step.

3. Which statement is true concerning the execution phase of the DATA step?
- a. Data is processed in the program data vector (PDV).
  - b. An implied OUTPUT occurs at the top of the DATA step.
  - c. An implied REINITIALIZE occurs at the bottom of the DATA step.
  - d. Columns read from the input table are set to missing when SAS returns to the top of the DATA step.

4. The DATA step debugger in SAS Enterprise Guide can be used with DATA and PROC steps.
  - a. True
  - b. False

4. The DATA step debugger in SAS Enterprise Guide can be used with DATA and PROC steps.
- a. True
- b. False

5. Which PUTLOG statements create the following results in the SAS log?

```
Name=Alfred Height=69 Weight=112.5 Ratio=0.61 _ERROR_=0 _N_=1  
Ratio=0.61
```

- a. **putlog all; putlog Ratio;**
- b. **putlog all; putlog Ratio=;**
- c. **putlog \_all\_; putlog Ratio;**
- d. **putlog \_all\_; putlog Ratio=;**

5. Which PUTLOG statements create the following results in the SAS log?

```
Name=Alfred Height=69 Weight=112.5 Ratio=0.61 _ERROR_=0 _N_=1  
Ratio=0.61
```

- a. `putlog all; putlog Ratio;`
- b. `putlog all; putlog Ratio=;`
- c. `putlog _all_; putlog Ratio;`
- d. `putlog _all_; putlog Ratio=;`

1. Which statement is false concerning the options for the PROC EXPORT statement?
  - a. The DATA= option identifies the input SAS table.
  - b. The REPLACE option specifies to overwrite an existing file.
  - c. The DBMS= option specifies the database identifier for the type of file being created.
  - d. The OUT= option specifies the path and file name of the external data file being created.

1. Which statement is false concerning the options for the PROC EXPORT statement?
  - a. The DATA= option identifies the input SAS table.
  - b. The REPLACE option specifies to overwrite an existing file.
  - c. The DBMS= option specifies the database identifier for the type of file being created.
  - d. The OUT= option specifies the path and file name of the external data file being created.

2. Which PROC EXPORT step contains valid syntax?

- a. 

```
proc export outfile="c:\temp\cars.txt" tab  
      data=sashelp.cars replace; run;
```
- b. 

```
proc export data=sashelp.cars dbms=csv  
      outfile="c:\temp\cars.csv"; run;
```
- c. 

```
proc export data=sashelp.class; dbms=csv;  
      outfile="c:\temp\cars.csv"; run;
```
- d. 

```
proc export dbms=tab data=sashelp.cars replace=yes  
      outfile="c:\temp\cars.txt"; run;
```

2. Which PROC EXPORT step contains valid syntax?

a.

```
proc export outfile="c:\temp\cars.txt" tab  
    data=sashelp.cars replace; run;
```

b.

```
proc export data=sashelp.cars dbms=csv  
    outfile="c:\temp\cars.csv"; run;
```

c.

```
proc export data=sashelp.class; dbms=csv;  
    outfile="c:\temp\cars.csv"; run;
```

d.

```
proc export dbms=tab data=sashelp.cars replace=yes  
    outfile="c:\temp\cars.txt"; run;
```

3. What does the following program create?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';

data sales.q1_2018;
    set sasdata.qtr1_2018;
run;
data sales.q2_2018;
    set sasdata.qtr2_2018;
run;
```

- a. two SAS tables: `sales.q1_2018` and `sales.q2_2018`
- b. two Excel workbooks: `sales.q1_2018` and `sales.q2_2018`
- c. two worksheets in the Excel workbook: `midyear: q1_2018` and `q2_2018`
- d. two worksheets in the Excel workbook: `sales: q1_2018` and `q2_2018`

3. What does the following program create?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';

data sales.q1_2018;
    set sasdata.qtr1_2018;
run;
data sales.q2_2018;
    set sasdata.qtr2_2018;
run;
```

- a. two SAS tables: `sales.q1_2018` and `sales.q2_2018`
- b. two Excel workbooks: `sales.q1_2018` and `sales.q2_2018`
- c. two worksheets in the Excel workbook: `midyear: q1_2018` and `q2_2018`
- d. two worksheets in the Excel workbook: `sales: q1_2018` and `q2_2018`

5. What type of output file does this program create?

```
libname mylib xlsx "s:/workshop/output/test.xlsx";  
  
data class_list;  
    set sashelp.class;  
run;
```

- a. SAS table
- b. delimited file
- c. Microsoft Excel XLS file
- d. Microsoft Excel XLSX file

5. What type of output file does this program create?

```
libname mylib xlsx "s:/workshop/output/test.xlsx";  
  
data class_list;  
    set sashelp.class;  
run;
```

- a. SAS table
- b. delimited file
- c. Microsoft Excel XLS file
- d. Microsoft Excel XLSX file

6. Which of these programs creates a Microsoft Excel file?

- a. 

```
ods excel file="s:/workshop/output/class.xlsx";
proc print data=sashelp.class;
run;
ods excel close;
```
- b. 

```
libname mylib xlsx "s:/workshop/output/class.xlsx";
data mylib.class_list;
  set sashelp.class;
run;
```
- c. both
- d. neither

6. Which of these programs creates a Microsoft Excel file?

a.

```
ods excel file="s:/workshop/output/class.xlsx";
proc print data=sashelp.class;
run;
ods excel close;
```

b.

```
libname mylib xlsx "s:/workshop/output/class.xlsx";
data mylib.class_list;
  set sashelp.class;
run;
```

- c. both
- d. neither

# SAS Lesson 06

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# Efficiently Changing the Filter Value

```
proc print data=sashelp.cars;  
  where Type="Wagon";  
  var Type Make Model MSRP;  
run;  
proc means data=sashelp.cars;  
  where Type="Wagon";  
  var MSRP MPG_Highway;  
run;  
proc freq data=sashelp.cars;  
  where Type="Wagon";  
  tables Origin Make;  
run;
```

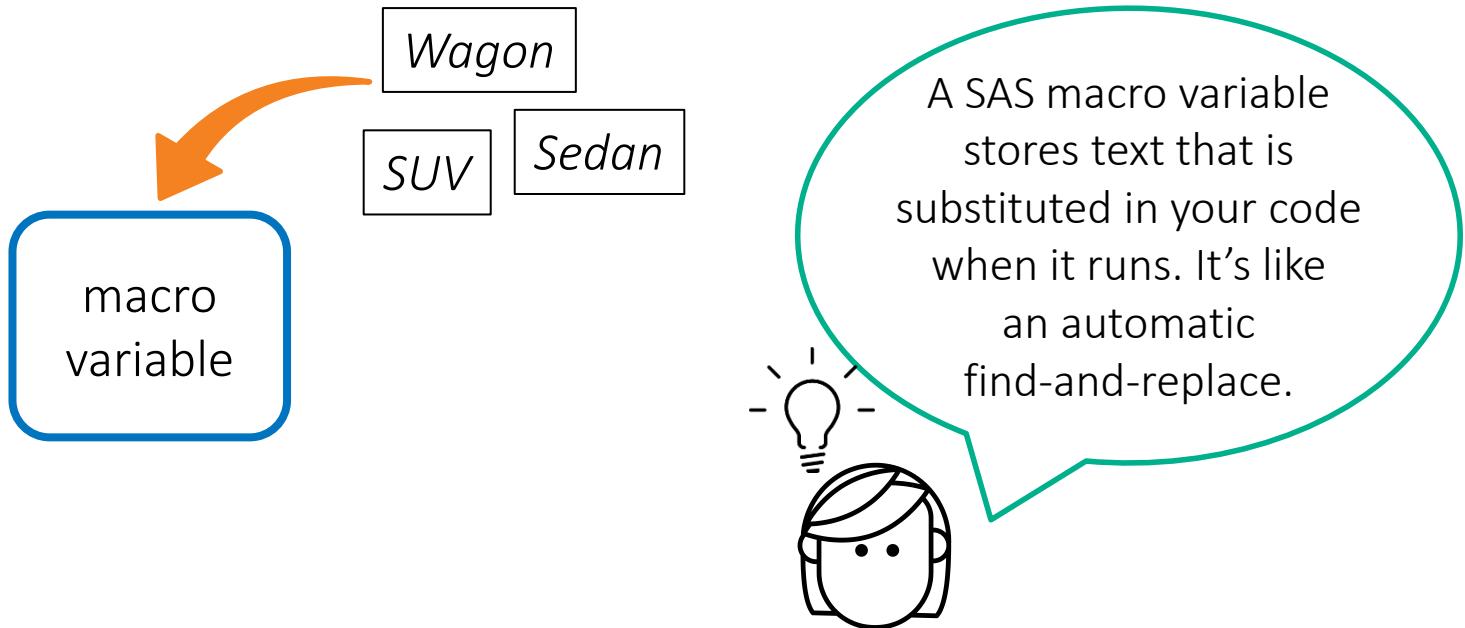
Wagon ⇌ SUV

```
proc print data=sashelp.cars;  
  where Type="SUV";  
  var Type Make Model MSRP;  
run;  
proc means data=sashelp.cars;  
  where Type="SUV";  
  var MSRP MPG_Highway;  
run;  
proc freq data=sashelp.cars;  
  where Type="SUV";  
  tables Origin Make;  
run;
```

How can you  
easily replace this  
value everywhere  
in the program?



# Efficiently Changing the Filter Value



# ~~A~~ Creating and Using SAS Macro Variables

create  
the  
macro  
variable

```
%let CarType=Wagon;
```

```
proc print data=sashelp.cars;  
    where Type="Wagon";  
    var Type Make Model MSRP;  
run;  
  
proc means data=sashelp.cars;  
    where Type="Wagon";  
    var MSRP MPG_Highway;  
run;  
  
proc freq data=sashelp.cars;  
    where Type="Wagon";  
    tables Origin Make;  
run;
```

```
%LET macro-variable=value;
```

creates a macro variable named `CarType` that stores the text `Wagon`

# Creating and Using SAS Macro Variables

use the  
macro  
variable

```
%let CarType=Wagon;  
  
proc print data=sashelp.cars;  
  where Type="&CarType";  
  var Type Make Model MSRP;  
run;  
proc means data=sashelp.cars;  
  where Type="&CarType";  
  var MSRP MPG_Highway;  
run;  
proc freq data=sashelp.cars;  
  where Type="&CarType";  
  tables Origin Make;  
run;
```

&macro-var

Use the macro variable  
in place of the value in  
the program.

# Creating and Using SAS Macro Variables

use the  
macro  
variable

```
%let CarType=Wagon;  
  
proc print data=sashelp.cars;  
  where Type="Wagon";  
  var Type Make Model MSRP;  
run;  
proc means data=sashelp.cars;  
  where Type="Wagon";  
  var MSRP MPG_Highway;  
run;  
proc freq data=sashelp.cars;  
  where Type="Wagon";  
  tables Origin Make;  
run;
```

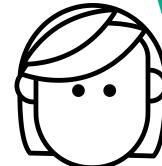


# Creating and Using SAS Macro Variables

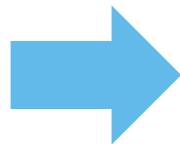
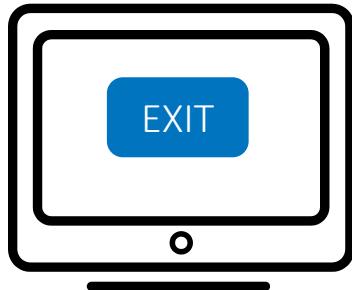
use the  
macro  
variable

```
%let CarType=SUV;  
  
proc print data=sashelp.cars;  
  where Type="SUV";  
  var Type Make Model MSRP;  
run;  
proc means data=sashelp.cars;  
  where Type="SUV";  
  var MSRP MPG_Highway;  
run;  
proc freq data=sashelp.cars;  
  where Type="SUV";  
  tables Origin Make;  
run;
```

You must change the value only in the %LET statement to change the filter value in all three procedures!

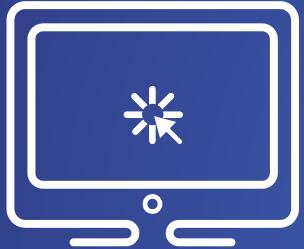


# Creating and Using SAS Macro Variables



Macro variables and  
their values are  
deleted when the  
SAS session ends.





# Filtering Rows Using Macro Variables

This demonstration illustrates modifying a program to use SAS macro variables to filter data in multiple steps.

# DATA Step Processing

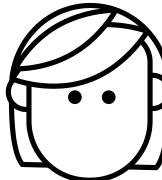
## Filtering Columns

# Subsetting Columns in the DATA Step

```
DROP col-name <col-name>;
```

```
KEEP col-name <col-name>;
```

Choose the statement based on the number of columns that you want to specify.



# Subsetting Columns in the DATA Step

These statements have the same result in the output table.

or

```
data myclass;  
  set sashelp.class;  
  keep name age height;  
  drop sex weight;  
run;
```

Name	Age	Height
Alfred	14	69
Alice	13	56.5
Barbara	13	65.3
Carol	14	62.8
Henry	14	63.5

# Preparing Data

Computing New Columns

# Using Expressions to Create New Columns

assignment statement

```
DATA output-table;  
  SET input-table;  
  new-column = expression;  
RUN;
```

arithmetic expression  
or constant

The assignment  
statement can  
create or update  
a column.

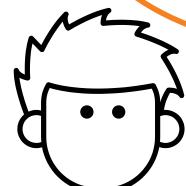


# Using Expressions to Create New Columns

```
data cars_new;  
  set sashelp.cars;  
  where Origin ne "USA";  
  Profit = MSRP-Invoice;  
  Source = "Non-US Cars";  
keep Make Model MSRP Invoice Profit Source;  
run;
```

Make	Model	MSRP	Invoice	Profit	Source
Acura	MDX	\$36,945	\$33,337	\$3,608	Non-US Cars
Acura	RSX Type S 2dr	\$23,820	\$21,761	\$2,059	Non-US Cars
Acura	TSX 4dr	\$26,990	\$24,647	\$2,343	Non-US Cars
Acura	TL 4dr	\$33,195	\$30,299	\$2,896	Non-US Cars
Acura	3.5 RL 4dr	\$43,755	\$39,014	\$4,741	Non-US Cars
Acura	3.5 RL w/Navi...	\$46,100	\$41,100	\$5,000	Non-US Cars
Acura	NSX coupe 2d...	\$89,765	\$79,978	\$9,787	Non-US Cars

The column name  
is stored in the  
case that you use  
to create it.



# Operands

Operands are constants (character, numeric, or date) and variables (character or numeric).

Examples:

**Bonus = 500;** ← numeric constant

**Gender = 'M';** ← character constant

**Hire\_Date = '01APR2008'd;** ← date constant

**NewSalary = 1.1 \* Salary;** ← variable

# Two Digit Years

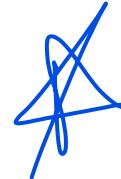
What happens if you enter this:

```
Payoff_Date = '01APR20'd; ← date constant
```

Obs	payoff_date
1	22006

## YEARCUTOFF

- System option that specifies first year of 100 year span for interpreting two-digit years
- Default in 9.4 = **1926**
- Use four-digit years to avoid misinterpretation



# Operators

Operators are symbols that represent an arithmetic calculation and SAS functions.

Examples:

```
Revenue = Quantity * Price;
```

```
NewCountry = upcase(Country) ;
```

# Arithmetic Operators

*Arithmetic operators* indicate that an arithmetic calculation is performed.

Symbol	Definition	Priority	
**	Exponentiation	1	
-	negative prefix	1	right to left within group 1
*	multiplication	2	
/	division	2	left to right within same priority group
+	addition	3	
-	subtraction	3	

- ☞ If a missing value is an operand for an arithmetic operator, the result is a missing value.

# Poll

# Quiz



# Quiz

What is the result of the assignment statement?

- a. . (missing)
- b. 0
- c. 7
- d. 9

```
num = 4 + 10 / 2;
```

# Quiz – Correct Answer

What is the result of the assignment statement?

- a. . (missing)
- b. 0
- c. 7
- d. 9

```
num = 4 + 10 / 2;
```

**The order of operations from left to right is division and multiplication followed by addition and subtraction.**

**Parentheses can be used to control the order of operations.**

```
num = (4 + 10) / 2;
```

# Quiz

What is the result of the assignment statement given the values of **var1** and **var2**?

- a. . (missing)
- b. 0
- c. 5
- d. 10

```
num = var1 + var2 / 2;
```

var1	var2
.	10

# Quiz – Correct Answer

What is the result of the assignment statement given the values of **var1** and **var2**?

- a. . (missing)
- b. 0
- c. 5
- d. 10

```
num = var1 + var2 / 2;
```

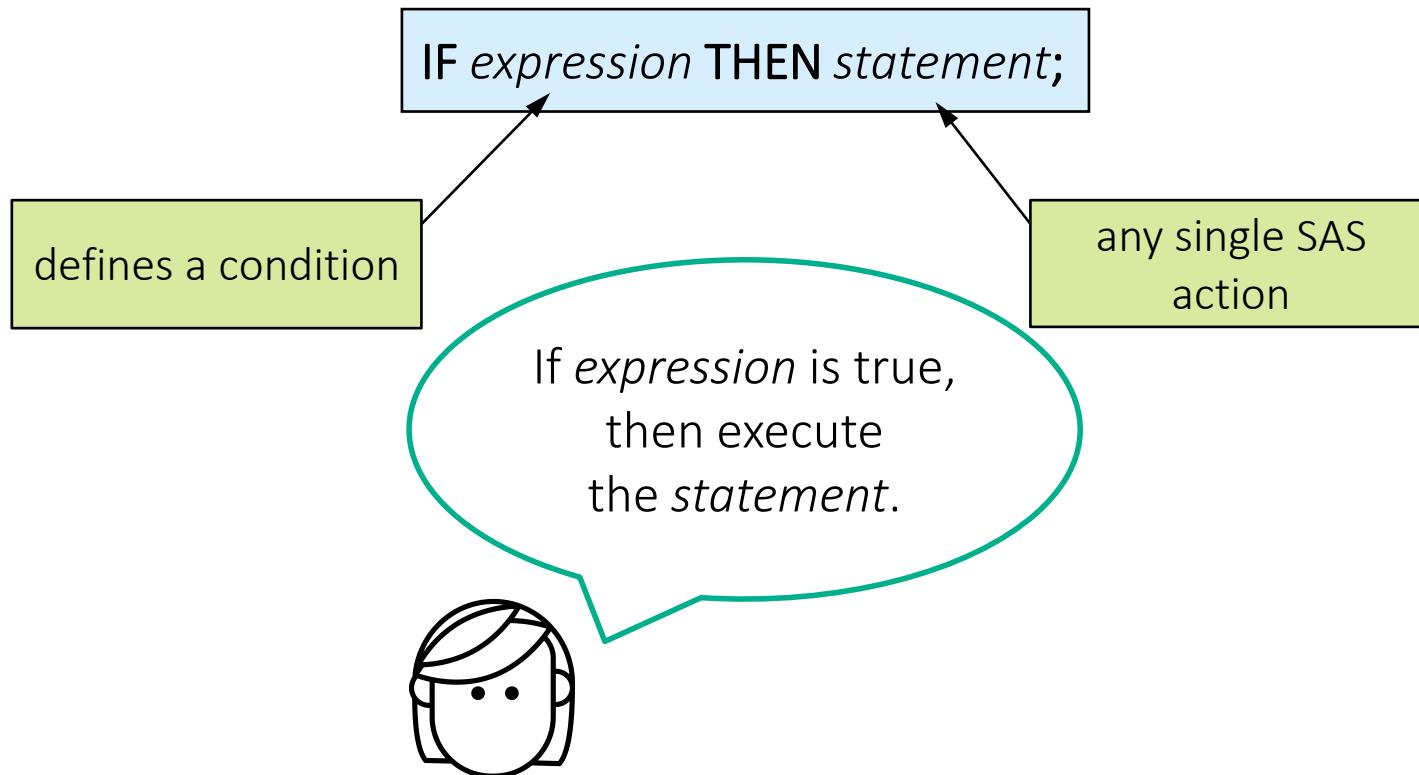
var1	var2
.	10

If an operand is missing for an arithmetic operator, the result is missing.

# Preparing Data

## Conditional Processing

# Conditional Processing with IF-THEN



# Conditional Processing with IF-THEN

```
data cars2;  
  set sashelp.cars;  
  if MSRP<30000 then Cost_Group=1;  
  if MSRP>=30000 then Cost_Group=2;  
  keep Make Model Type MSRP Cost_Group;  
run;
```

Make	Model	Type	MSRP	Cost_Group
Acura	MDX	SUV	\$36,945	2
Acura	RSX Type S 2dr	Sedan	\$23,820	1
Acura	TSX 4dr	Sedan	\$26,990	1
Acura	TL 4dr	Sedan	\$33,195	2
Acura	3.5 RL 4dr	Sedan	\$43,755	2
Acura	3.5 RL w/Navi...	Sedan	\$46,100	2
Acura	NSX coupe 2d...	Sports	\$89,765	2
Audi	A4 1.8T 4dr	Sedan	\$25,940	1

# Conditional Test for Missing

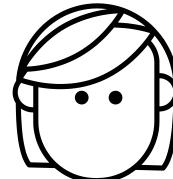
Expression	What It Does
$N = .$	Returns TRUE if numeric value is missing
$C = ''$	Returns TRUE if character value is missing
MISSING ( <i>variable</i> )	Returns TRUE if the numeric or character argument is missing

# Conditional Processing with IF-THEN/ELSE

```
IF expression THEN statement;  
<ELSE IF expression THEN statement;>  
<ELSE IF expression THEN statement;>  
ELSE statement;
```

If the others are not true, execute this statement.

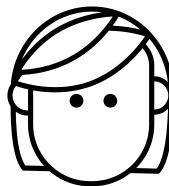
If *expression* is true,  
then execute  
the *statement* and  
skip the rest.



# Conditional Processing with IF-THEN/ELSE

```
data cars2;  
    set sashelp.cars;  
    if MSRP<20000 then Cost_Group=1;  
    else if MSRP<40000 then Cost_Group=2;  
    else if MSRP<60000 then Cost_Group=3;  
    else Cost_Group=4;  
    keep Make Model Type MSRP Cost_Group;  
run;
```

For efficiency, place  
the most frequently  
occurring conditions at  
the top.



# Conditional Processing with IF-THEN/ELSE

Example: MSRP=35000

```
data cars2;
  set sashelp.cars;
  if MSRP<20000 then Cost_Group=1;
  else if MSRP<40000 then Cost_Group=2;
  else if MSRP<60000 then Cost_Group=3;
  else Cost_Group=4;
  keep Make Model Type MSRP Cost_Group;
run;
```

false

```
if MSRP<20000 then Cost_Group=1;
```

```
else if MSRP<40000 then Cost_Group=2;
```

```
else if MSRP<60000 then Cost_Group=3;
```

```
else Cost_Group=4;
```

```
keep Make Model Type MSRP Cost_Group;
```

# Conditional Processing with IF-THEN/ELSE

Example: MSRP=35000

```
data cars2;  
    set sashelp.cars;  
    if MSRP<20000 then Cost_Group=1;  
    else if MSRP<40000 then Cost_Group=2;  
    else if MSRP<60000 then Cost_Group=3;  
    else Cost_Group=4;  
    keep Make Model Type MSRP Cost_Group;  
run;
```

true

execute

# Conditional Processing with IF-THEN/ELSE

Example: MSRP=35000

```
data cars2;
  set sashelp.cars;
  if MSRP<20000 then Cost_Group=1;
  else if MSRP<40000 then Cost_Group=2;
  else if MSRP<60000 then Cost_Group=3;
  else Cost_Group=4;
  keep Make Model Type MSRP Cost_Group;
run;
```

skip

# Conditional Processing with IF-THEN/ELSE

Example: MSRP=75000

```
data cars2;  
    set sashelp.cars;  
    if MSRP<20000 then Cost_Group=1;  
    else if MSRP<40000 then Cost_Group=2;  
    else if MSRP<60000 then Cost_Group=3;  
    else Cost_Group=4;  
    keep Make Model Type MSRP Cost_Group;  
run;
```

false

execute

The final ELSE statement executes if all previous conditions were false.

# Creating Character Columns with IF-THEN/ELSE

```
data cars2;  
  set sashelp.cars;  
  if MSRP<60000 then CarType="Basic";  
  else CarType="Luxury";  
  keep Make Model MSRP CarType;  
run;
```

Based on the value of MSRP,  
assign a value to the new  
character column CarType.

Make	Model	MSRP	CarType
Acura	MDX	\$36,945	Basic
Acura	RSX Type S 2dr	\$23,820	Basic
Acura	TSX 4dr	\$26,990	Basic
Acura	TL 4dr	\$33,195	Basic
Acura	3.5 RL 4dr	\$43,755	Basic
Acura	3.5 RL w/Navi...	\$46,100	Basic
Acura	NSX coupe 2d...	\$89,765	Luxur
Audi	A4 1.8T 4dr	\$25,940	Basic

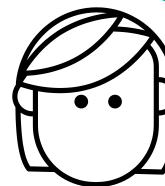
# Creating Character Columns with IF-THEN/ELSE

```
data cars2;  
  set sashelp.cars;  
  if MSRP<60000 then CarType="Basic";  
  else CarType="Luxury";  
  keep Make Model MSRP CarType;  
run;
```

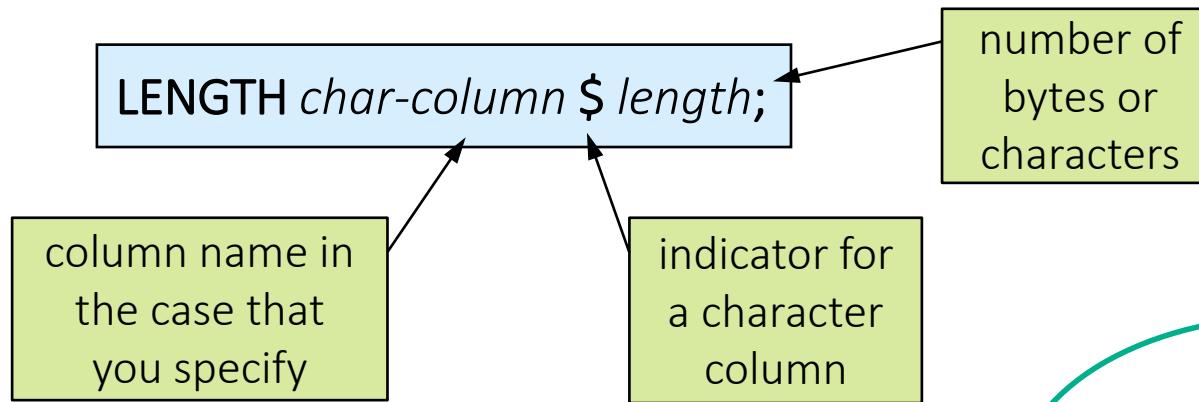
creates a new character column with a length of 5

Make	Model	MSRP	CarType
Acura	MDX	\$36,945	Basic
Acura	RSX Type S 2dr	\$23,820	Basic
Acura	TSX 4dr	\$26,990	Basic
Acura	TL 4dr	\$33,195	Basic
Acura	3.5 RL 4dr	\$43,755	Basic
Acura	3.5 RL w/Navi...	\$46,100	Basic
Acura	NSX coupe 2d...	\$89,765	Luxur
Audi	A4 1.8T 4dr	\$25,940	Basic

The first mention of a column in the DATA step defines the name, type, and length.



# Creating Character Columns with IF-THEN/ELSE



Using the LENGTH statement leaves nothing to chance.



# Creating Character Columns with IF-THEN/ELSE

```
data cars2;  
  set sashelp.cars;  
  length CarType $ 6;  
  if MSRP<60000 then CarType="Basic";  
  else CarType="Luxury";  
  keep Make Model MSRP CarType;  
run;
```

explicitly creates  
a new character column  
with a length of 6

⚠ Make	⚠ Model	💵 MSRP	⚠ CarType
Acura	MDX	\$36,945	Basic
Acura	RSX Type S 2dr	\$23,820	Basic
Acura	TSX 4dr	\$26,990	Basic
Acura	TL 4dr	\$33,195	Basic
Acura	3.5 RL 4dr	\$43,755	Basic
Acura	3.5 RL w/Navi...	\$46,100	Basic
Acura	NSX coupe 2d...	\$89,765	Luxury
Audi	A4 1.8T 4dr	\$25,940	Basic



# Conditional Processing with IF-THEN

This demonstration illustrates using IF-THEN syntax to assign values conditionally to a new column.

# Using Compound Conditions with IF-THEN/ELSE

```
data cars2;  
  set sashelp.cars;  
  if MPG_City>26 and MPG_Highway>30 then Efficiency=1;  
  else if MPG_City>20 and MPG_Highway>25 then Efficiency=2;  
  else Efficiency=3;  
  keep Make Model MPG_City MPG_Highway Efficiency;  
run;
```

AND

Both conditions  
must be true.

OR

One condition  
must be true.

⚠ Make	⚠ Model	⌚ MPG_City	⌚ MPG_Highway	⌚ Efficiency
Chevrolet	Tracker	19	22	3
Chevrolet	Aveo 4dr	28	34	1
Chevrolet	Aveo LS 4dr hatch	28	34	1
Chevrolet	Cavalier 2dr	26	37	2
Chevrolet	Cavalier 4dr	26	37	2

# Processing Multiple Statements

*Compound statements not allowed in SAS.*

```
data cars2;  
  set sashelp.cars;  
  length Cost_Type $ 4;  
  if MSRP<20000 then Cost_Group=1 and Cost_Type="Low";  
  else if MSRP<40000 then Cost_Group=2 and Cost_Type="Mid";  
  else Cost_Group=3 and Cost_Type="High";  
run;
```

Compound statements are not allowed.

This program doesn't work because only one statement is permitted after THEN.



# Processing Multiple Statements with IF-THEN/DO

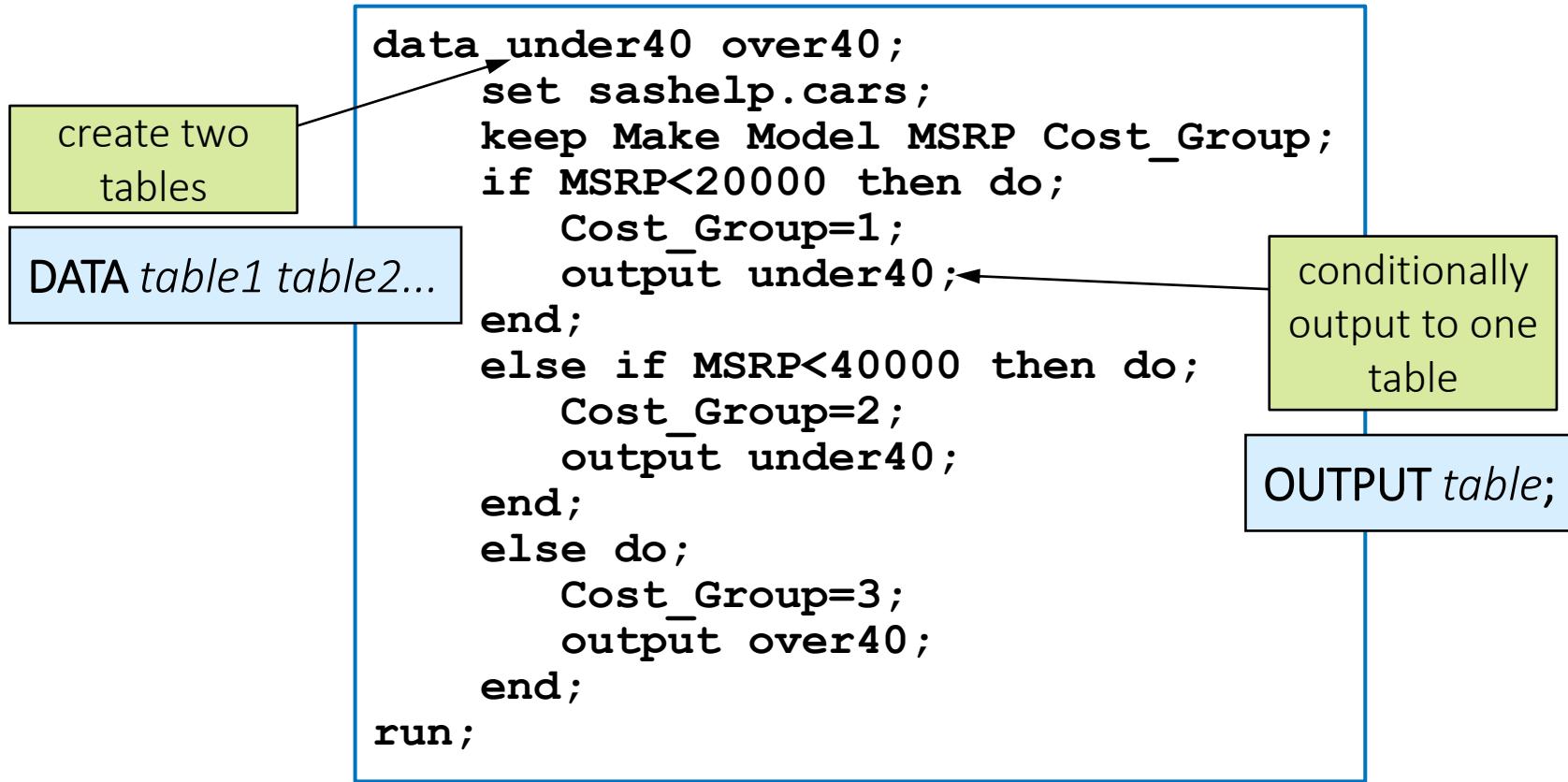
```
IF expression THEN DO;  
  <executable statements>  
END;  
ELSE IF expression THEN DO;  
  <executable statements>  
END;  
ELSE DO;  
  <executable statements>  
END;
```

Well, END says it's over.  
For DO block.

If *expression* is true,  
then execute all the  
*statements* between  
DO and END.



# Processing Multiple Statements with IF-THEN/DO



# QUIZ

Why does the program fail?

```
data girls boys;
  set sashelp.class;
  if sex="F" then do;
    Gender="Female";
    output girls;
  else do;
    Gender="Male";
    output boys;
run;
```

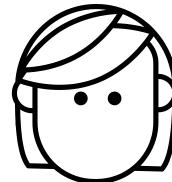
# QUIZ– Correct Answer

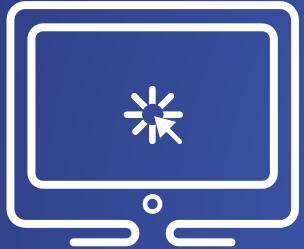
Why does the program fail?

ERROR 117-185: There were 2 unclosed DO blocks.

```
data girls boys;
  set sashelp.class;
  if sex="F" then do;
    Gender="Female";
    output girls;
  end;
  else do;
    Gender="Male";
    output boys;
  end;
run;
```

Using the Format Code feature in Enterprise Guide and SAS Studio helps you identify the DO blocks.





# Processing Multiple Statements with IF-THEN/DO

This demonstration illustrates using  
IF-THEN/DO syntax to execute multiple  
statements for each condition.

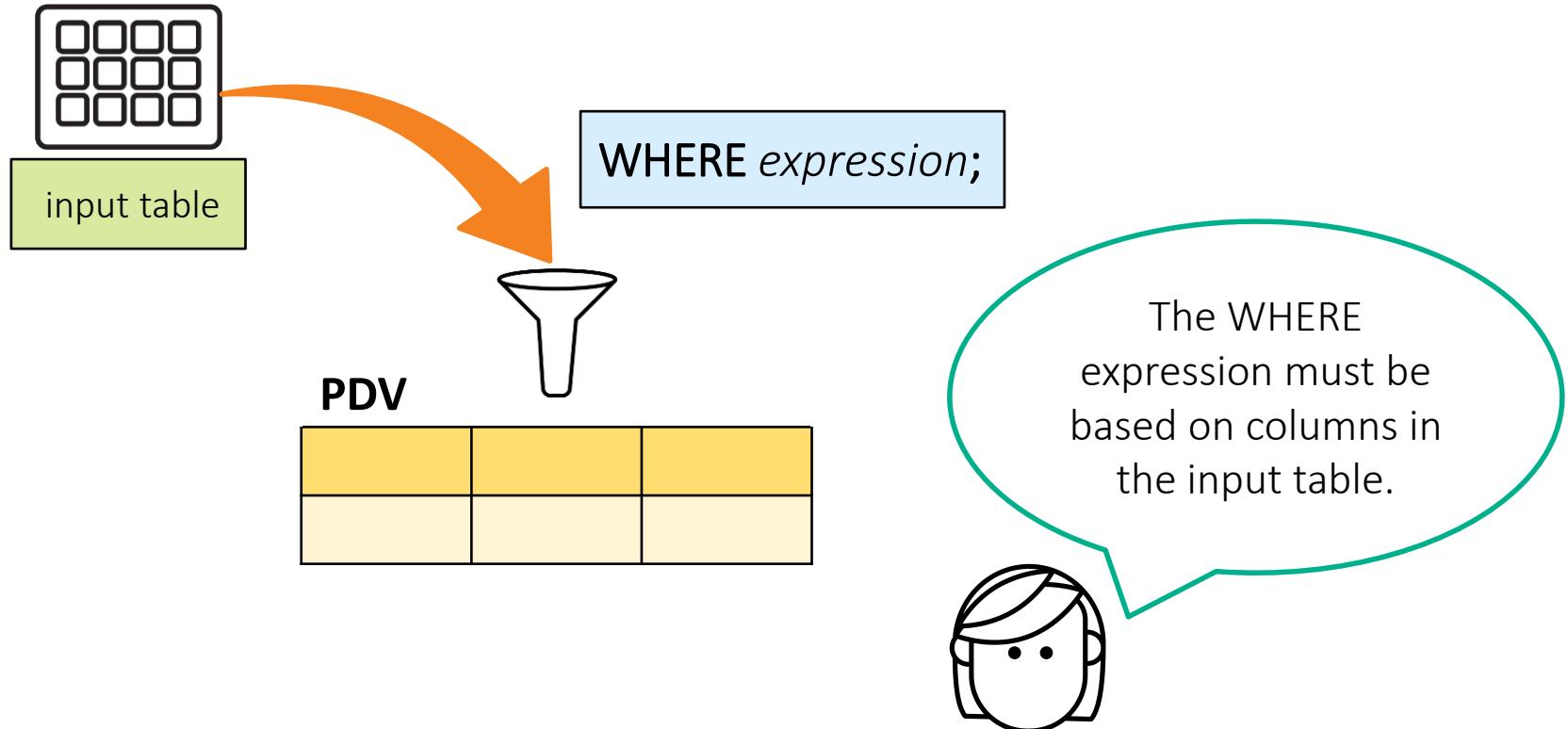
# Subsetting Rows in Execution

true

```
data cheapcars;  
  set sashelp.cars;  
  if MSRP < 30000;  
run;
```

What action is taken here  
when the IF condition is true?

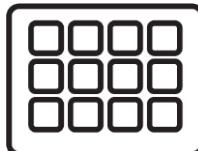
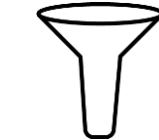
# Subsetting Rows in Execution



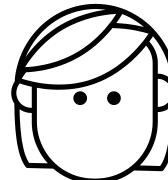
# Subsetting Rows in Execution

PDV


IF expression;



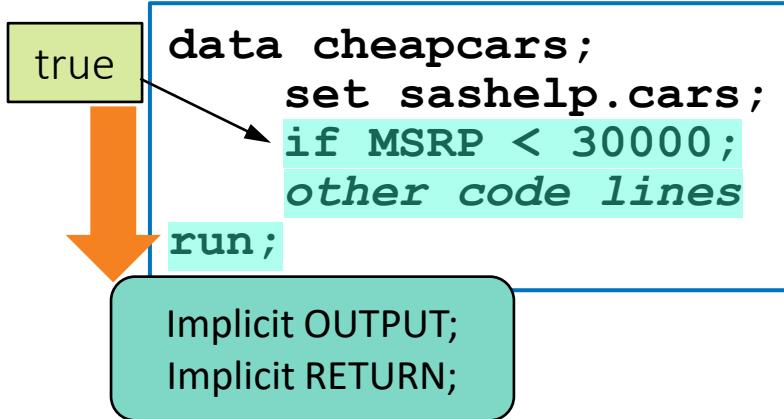
output table



The IF expression  
can be based on any  
values in the PDV.

(IN) THE  
USED IN  
CLASSICAL  
WORKS  
in DATA  
STEP,

# Subsetting Rows in Execution



When the IF condition is true,  
SAS continues processing  
statements for that row.

# Subsetting in Execution

Implicit RETURN;

```
data cheapcars;  
  set sashelp.cars;  
  if MSRP < 30000;  
  other code lines  
run;
```

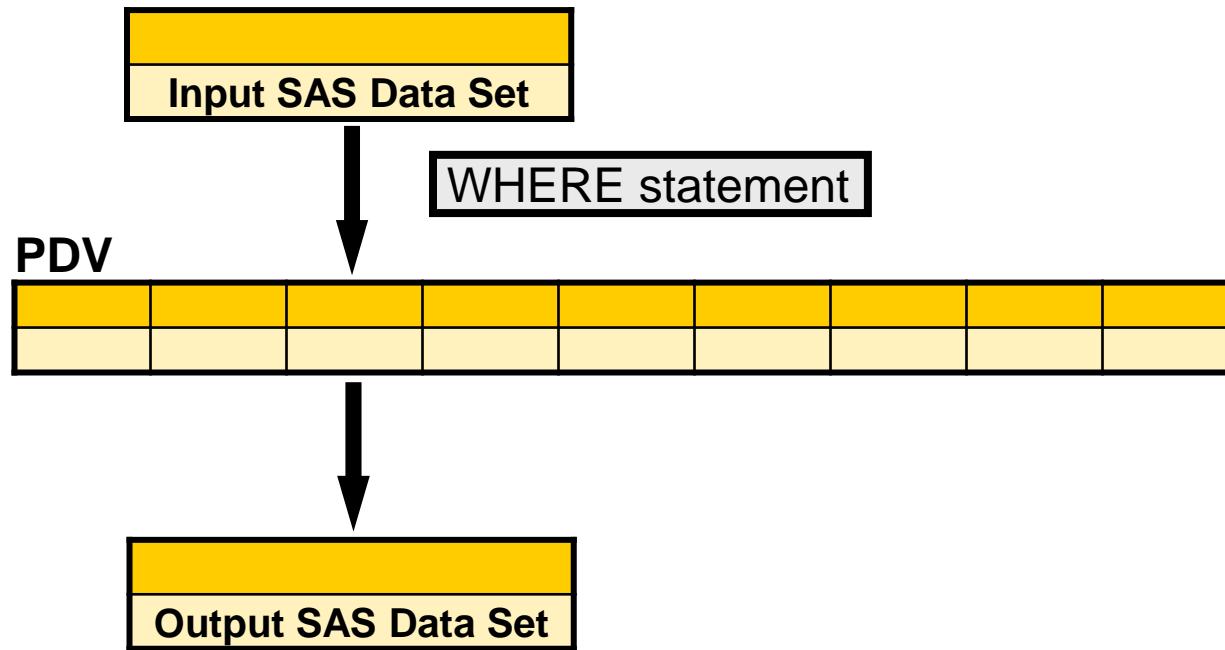
Implicit OUTPUT;

false

When the IF condition is false,  
SAS stops processing statements  
for that row and returns to the  
top of the DATA step.

# Processing the WHERE Statement

The WHERE statement selects observations **before** they are brought into the program data vector.



# Quiz

Why does the WHERE statement not work in this DATA step?

```
data work.december;
  set orion.sales;
  BonusMonth=month(Hire_Date);
  Bonus=500;
  Compensation=sum(Salary,Bonus);
  where Country='AU' and BonusMonth=12;
run;
```

# Quiz – Correct Answer

Why does the WHERE statement not work in this DATA step?

```
data work.december;
  set orion.sales;
  BonusMonth=month(Hire_Date);
  Bonus=500;
  Compensation=sum(Salary,Bonus);
  where Country='AU' and BonusMonth=12;
run;
```

ERROR: Variable BonusMonth is not on file ORION.SALES.

The WHERE statement can only subset variables that are coming from an existing data set.

# The Subsetting IF Statement

Examples:

```
if Salary > 50000;
```

```
if Last_Name='Smith' and First_Name='Joe';
```

```
if Country not in ('GB', 'FR', 'NL');
```

```
if Hire_Date = '15APR2008'd;
```

```
if not missing(end_date);
```

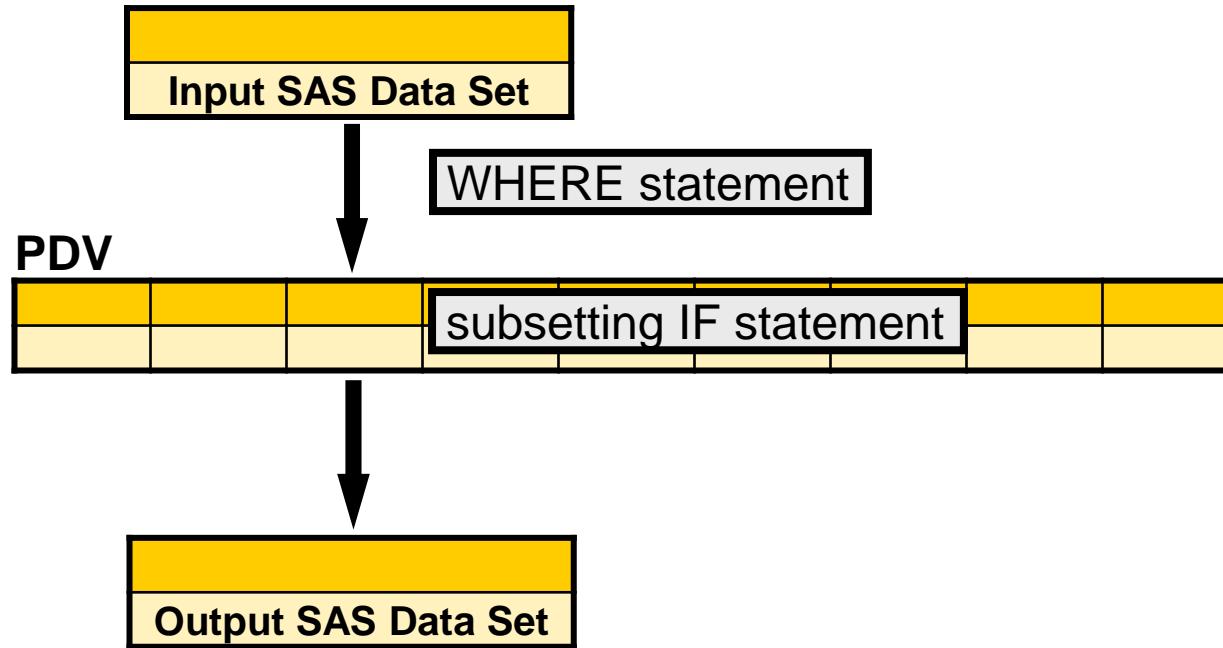
```
if upcase(Gender)='M';
```

```
if 40000 <= Compensation <= 80000;
```

```
if Salary+Bonus < 43000;
```

# Processing the Subsetting IF Statement

The subsetting IF statement determines if observations continue being processed in the program data vector.



# Business Scenario

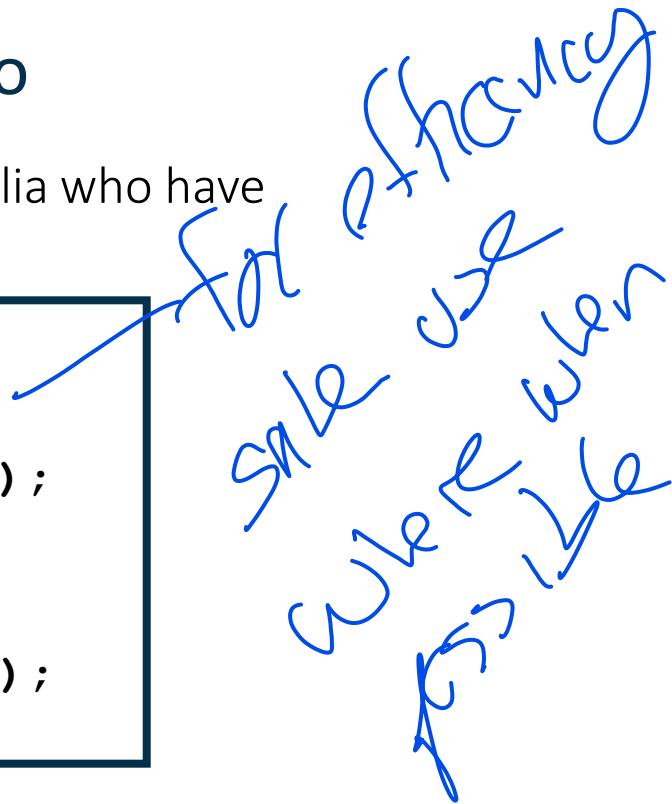
Include only the employees from Australia who have a bonus month in December.

```
data work.december;
  set orion.sales;
  where Country='AU';
  BonusMonth=month(Hire_Date);
  if BonusMonth=12;
  Bonus=500;
```

```
Compensation=sum(Salary,Bonus);
run;
```

Partial SAS Log

```
NOTE: There were 63 observations read from the data set ORION.SALES.
      WHERE Country='AU';
NOTE: The data set WORK.DECEMBER has 3 observations and 12 variables.
```



# SAS Lesson 07

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# Business Scenario

Include only the employees from Australia who have a bonus month in December.

```
data work.december;
  set orion.sales;
  where Country='AU';
  BonusMonth=month(Hire_Date);
  if BonusMonth=12;
  Bonus=500;

  Compensation=sum(Salary,Bonus);
run;
```

Partial SAS Log

```
NOTE: There were 63 observations read from the data set ORION.SALES.
      WHERE Country='AU';
NOTE: The data set WORK.DECEMBER has 3 observations and 12 variables.
```

# Quiz

Could you write only an IF statement?

- Yes
- No

```
data work.december;
  set orion.sales;
  where Country='AU';
  BonusMonth=month(Hire_Date);
  if BonusMonth=12;
  Bonus=500;
  Compensation=0;
run;
```

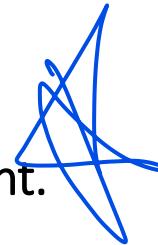
```
data work.december;
  set orion.sales;
  BonusMonth=month(Hire_Date);
  if BonusMonth=12 and Country='AU';
  Bonus=500;
  Compensation=sum(Salary,Bonus);
run;
```

# Quiz – Correct Answer

Could you write only an IF statement?

- Yes
- No

Yes, but the program using both the WHERE and IF statements is more efficient.



Both methods create a data set with three observations. The program using both statements reads 63 observations into the PDV. The program using only the IF statement reads 165 observations into the PDV.

# WHERE Statement versus Subsetting IF Statement

Step and Usage	WHERE	IF
<b>PROC step (Except IMPORT)</b>	Yes	No
<b>DATA step (source of variable)</b>		
INPUT statement	No	Yes
assignment statement	No	Yes
SET statement (single data set)	Yes	Yes
SET/MERGE statement (multiple data sets)		
Variable in ALL data sets	Yes	Yes
Variable not in ALL data sets	No	Yes

# The IF-THEN DELETE Statement

An alternative to the subsetting IF statement is the DELETE statement in an IF-THEN statement.

General form of the IF-THEN DELETE statement:

**IF expression THEN DELETE;**

The *DELETE statement* stops processing the current observation.

# The IF-THEN DELETE Statement

```
data work.december;
  set orion.sales;
  where Country='AU';
  BonusMonth=month(Hire_Date);
  if BonusMonth ne 12 then delete;
  Bonus=500;
  Compensation=sum(Salary,Bonus);
run;
```

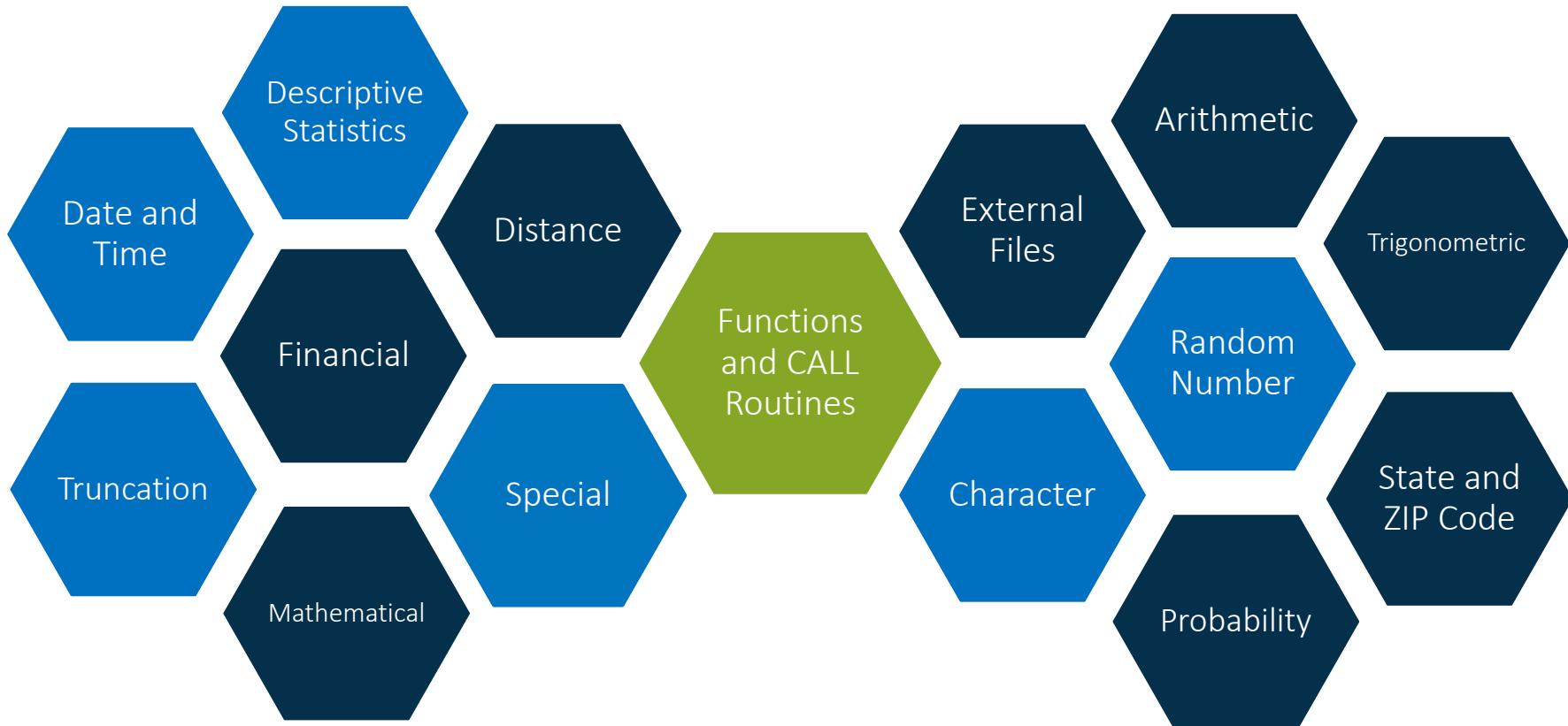
equivalent

```
data work.december;
  set orion.sales;
  where Country='AU';
  BonusMonth=month(Hire_Date);
  if BonusMonth=12;
  Bonus=500;
  Compensation=sum(Salary,Bonus);
run;
```

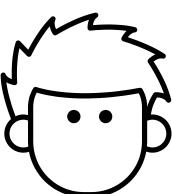
# Manipulating Data with Functions

Understanding SAS Functions and CALL Routines – Ch. 14

# SAS Functions and CALL Routines

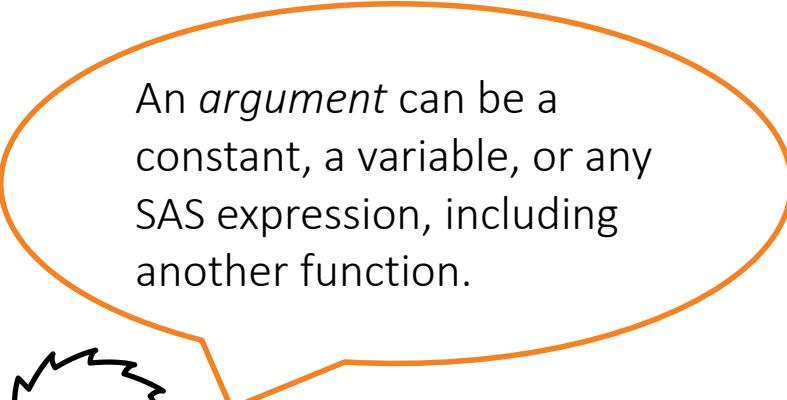


# What Is a SAS Function?

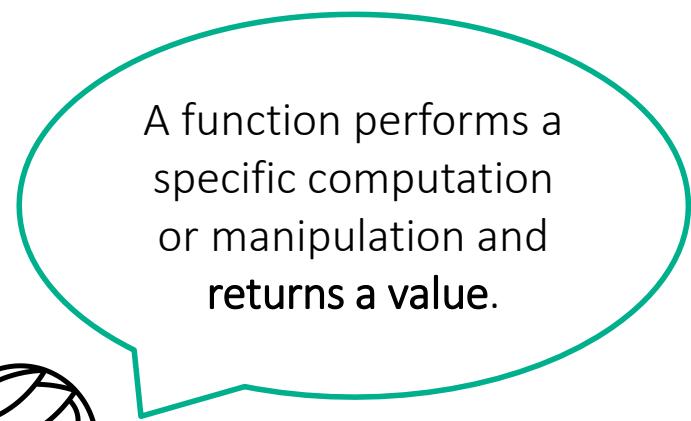


$f(\cdot)$

`function(argument1, argument2, ...);`



An *argument* can be a constant, a variable, or any SAS expression, including another function.



A function performs a specific computation or manipulation and **returns a value**.

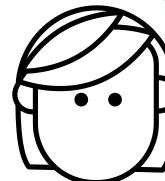
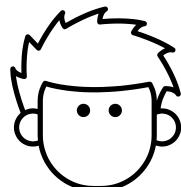
# Target Variables

```
Full_Name=cat (First_Name, ' ', Last_Name);
```

A target variable is the variable to which a function result is assigned.

Default lengths depend on the function and can be as long as 200 characters.

Add a LENGTH statement to avoid using unnecessary space.



# Using SAS Functions

You can use functions in DATA step statements anywhere that an expression can appear.

```
data contrib;
  set orion.employee_donations;
  Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
  if Total ge 50;
run;
```

Contributions \$50 and Over

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4	Total
120267	15	15	15	15	60
120269	20	20	20	20	80
120271	20	20	20	20	80
120275	15	15	15	15	60
120660	25	25	25	25	100

# SAS Variable Lists

An alternative to typing variables separately is to use a SAS *variable list*.

```
data contrib;
  set orion.employee_donations;
  Total=sum(of Qtr1-Qtr4);
  if Total ge 50;
run;
```

When you use a SAS variable list in a SAS function, use the keyword OF in front of the first variable name in the list.



# Quiz

What happens if you forget the keyword OF?

```
data contrib;
  set orion.employee_donations;
  Total=sum(Qtr1-Qtr4);
  if Total ge 50;
run;
```

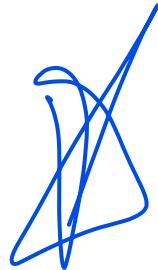
# Quiz – Correct Answer

What happens if you forget the keyword OF?

```
data contrib;
  set orion.employee_donations;
  Total=sum(Qtr1-Qtr4);
  if Total ge 50;
run;
```

QTR1-QTR4 will be interpreted  
as QTR1 minus QTR4.

# SAS Variable Lists



A SAS *variable list* is an abbreviated method of referring to a group of variable names. SAS enables you to use the following variable lists:

- Numbered range (X1-Xn)
- Name range (X--A, X-numeric-A, X-character-A)
- Name prefix (X:)
- Special SAS name lists, by variable type (\_ALL\_, \_NUMERIC\_, \_CHARACTER\_)

# SAS Variable Lists – Examples

PDV

*Numbered Range List*

Qtr1	Qtr2	Var1	Qtr3	Qtr4

Total = sum(of Qtr1-Qtr4)

Var1 not included

PDV

*Name Range List (by position in PDV)*

Qtr1	Second	Q3	Fourth	Var2

Total = sum(of Qtr1--Fourth)

Var2 not included

# SAS Variable Lists – Examples

PDV

*Name Prefix List*

TotJan	Qtr2	TotFeb	TotMar

`Total = sum(of Tot:)`

Qtr2 not included

PDV

*Special Name Lists*

Qtr1 N	Name \$	Q3 N	Fourth N

`Total = sum(of _Numeric_)`

Name not included

# Specifying Variable Lists

```
data quiz_summary;  
  set pg2.class_quiz;  
  Name=uppercase(Name);  
  AvgQuiz=mean(of Q:);  
  format Quiz1--AvgQuiz 3.1;  
  /*OR*/  
  format _numeric_ 3.1;  
  
run;
```

The keyword  
\_NUMERIC\_  
includes all  
numeric columns.

The double dash includes all  
columns between and  
including the two specified  
columns as they are ordered  
in the PDV.

The name prefix  
includes all columns  
whose name begins with  
Q.

Variable lists can  
be used in  
statements as  
well!



# Quiz

Complete the assignment statement for **Total** by using a SAS variable list and the SUM function to add the values for **Year1**, **Year2**, **Year3**, and **Year4**.

**PDV**

Year2	Year1	Year3	Year4	Sales

```
Total = sum(of ? )
```

# Correct Answer

Any of these assignment statements would give the correct value for **Total**.

**PDV**

Year2	Year1	Year3	Year4	Sales

```
Total = sum(of Year1-Year4) ;
```

```
Total = sum(of Year2--Year4) ;
```

```
Total = sum(of Year:) ;
```

# What Is a SAS CALL Routine?

`CALL routine(argument-1 <, ...argument-n>);`

A CALL routine is used in a CALL statement.

CALL routines **alter** column values or perform system **actions**. They **cannot** be used in assignment statements or expressions.



# Using a CALL Routine to Modify Data

```
data quiz_report;  
  set pg2.class_quiz;  
  if Name in("Barbara", "James")  
    then call missing(of Q:);  
run;
```

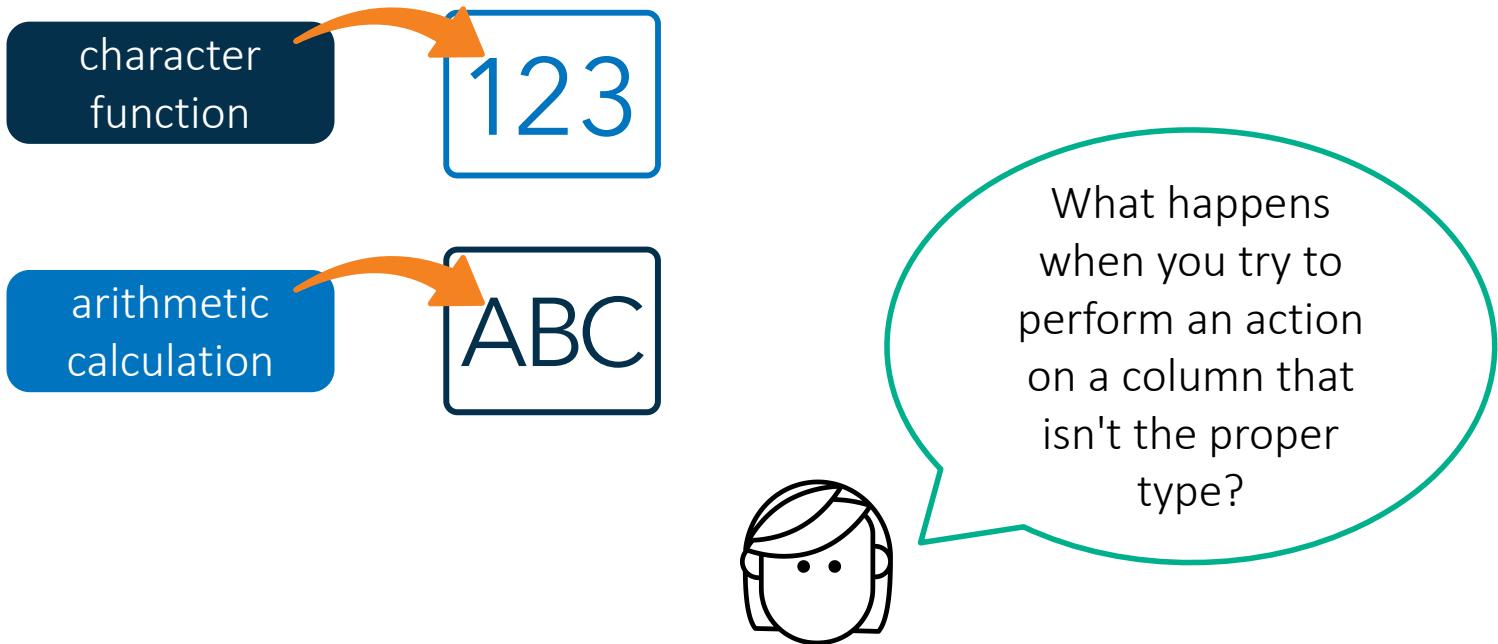
CALL MISSING assigns missing values to each variable in the argument list (character or numeric).

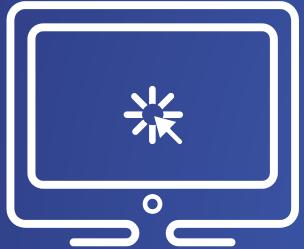
	Name	Quiz1	Quiz2	Quiz3	Quiz4	Quiz5
1	Alfred	8	7	6	9	8
2	Alice	7	6	4	9	8
3	Barbara	.	.	.	.	.
4	Carol	6	5	5	8	8
5	Henry	8	.	6	10	7
6	James	.	.	.	.	.
7	Jane	8	7	6	9	6

# Manipulating Data with Functions

Using Special Functions to Convert Column Type

# Handling Column Type





# Automatic Conversion

This demonstration illustrates the automatic conversion of values from character to numeric.

# Automatic Conversion of Column Type

**Range = High-Low;**

High	Low	Range
89.92	81.56	8.36
89.94	80.64	9.3
84.6	78.7	5.9
82.11	76.93	5.18
84.2	79.87	4.33

Automatic conversion is successful because **High** contains standard numeric values.

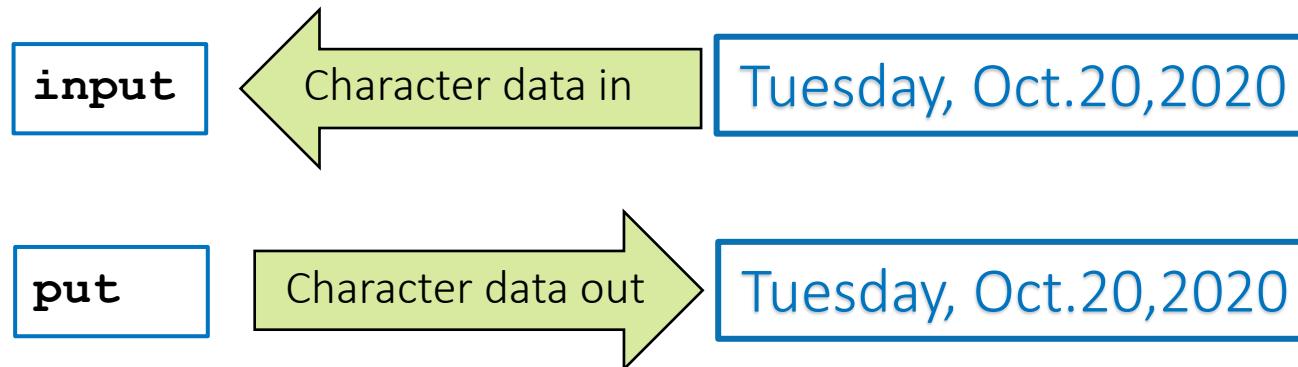
**DailyVol = Volume/30;**

Volume	DailyVol
5,025,627	.
4,455,657	.
5,960,945	.
5,043,800	.
5,754,876	.

Automatic conversion fails because **Volume** contains nonstandard numeric values.

# Conversion Functions

Function	What it does
<code>INPUT(source, informat)</code>	Converts character values to numeric or other character values using a specified informat
<code>PUT(source, format)</code>	Converts numeric or character values to character values using a specified format



# Converting Character Values to Numeric Values

character to numeric

Stock	Date	Open	Close
ABC Company	01DEC2017	89.15	82.2
ABC Company	01NOV2017	81.85	88.9
ABC Company	02OCT2017	80.22	81.88
ABC Company	01SEP2017	80.16	80.22

Stock	Date2	Open	Close
ABC Company	21154	89.15	82.2
ABC Company	21124	81.85	88.9
ABC Company	21094	80.22	81.88
ABC Company	21063	80.16	80.22

```
Date2=input(Date,date9.);
```

source

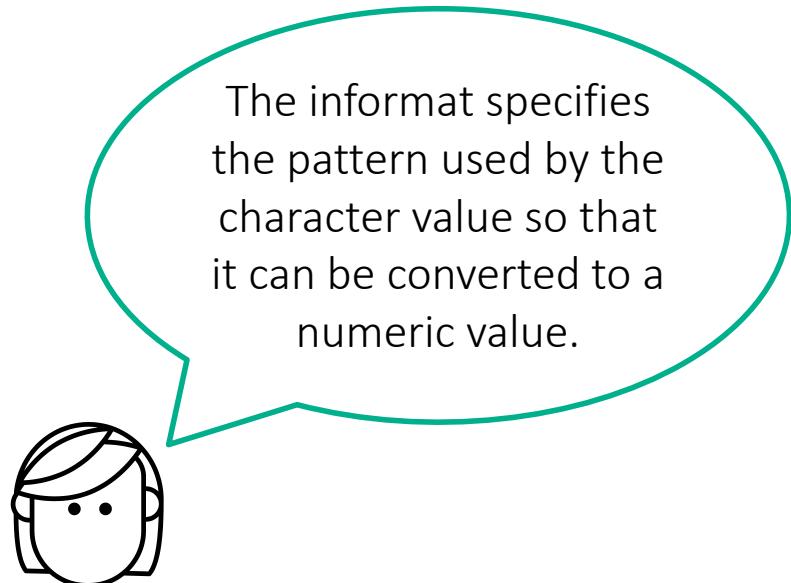
informat

# What is an informat?

- Predefined instruction
- Informs SAS how to interpret data
- Usually has a name
- Begins with \$ to indicate character informat
- Has a field width
- Must have a period delimiter
- Determines data type of variables created with input function

# Informats for Converting Character to Numeric

Character	Informat	Numeric
15OCT2018	DATE9.	21472
10/15/2018	MMDDYY10.	21472
15/10/2018	DDMMYY10.	21472
123,456.78	COMMA12.	123456.78
\$123,456.78	DOLLAR12.	
123456	6.	123456



The informat specifies the pattern used by the character value so that it can be converted to a numeric value.

# Date, Datetime, and Time Values

SAS date



SAS time



SAS datetime



# Additional Date and Time Informats

Character	Informat	Numeric
101518	MMDDYY6.	21472
6:45 PM	TIME8.	67500
18:45	TIME8.	67500
20Oct2020 6:45 PM	DATETIME18.	1918838700
20Oct2020 18:45:00	DATETIME18.	1918838700
10/20/2020 18:45:00	ANYDTDTE19.	22208
10/20/2020 18:45:00	ANYDTDTM19.	1918838700

# Informats for Converting Character to Numeric

Character
15OCT2018
10/15/2018
10152018
20181015
Oct 15, 2018
October 15, 2018

ANYDTDTEw.

ANYDTDTMw.

SAS Date

21472

21472

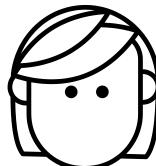
21472

21472

21472

21472

The ANYDTDTE  
and ANYDTDTM  
informats can  
read dates written  
in many ways.

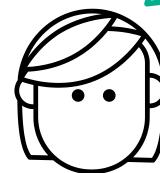


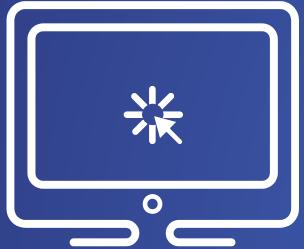
# Informats for Converting Character to Numeric

```
data work.stocks2;  
  set pg2.stocks2;  
  NewVolume1=input(Volume,comma12.);  
  NewVolume2=input(Volume,comma12.2);  
  keep volume newvolume:;  
run;
```

⚠ Volume	⑫ NewVolume1	⑬ NewVolume2
5,976,252	5976252	59762.52
5,556,471	5556471	55564.71
7,019,666	7019666	70196.66
5,772,280	5772280	57722.8

Be careful not to specify a decimal value with the informat unless you want to insert a new decimal point.





# Explicit Conversion

This demonstration illustrates the explicit conversion of date/time values from character to numeric.

# Facts about the PUT Function



- Always returns a character string
- Returns the source written with a format
- The format must be the same type as the source
- Numeric formats right align results
- Character formats left align results
- Length of new variables is equal to the format width

# What is a format?

- Controls the way data values are displayed or printed
- Usually has a name
- Begins with \$ to indicate character format ~~✓~~
- Width must be sufficient for data and all characters and decimal places
- Must have a period delimiter
- Permanently applied in DATA step
- Does NOT change the actual value unless *assigned* with a PUT function
- Temporarily applied in PROC steps

# Formatting Data Values in Results

data

ID	HireDate	Salary
120201	21020	62000
145675	20742	76000
184430	21000	35000

\$w.  
W.d

report

ID	HireDate	Salary
120201	01FEB2015	\$62,000
145675	15OCT2016	\$76,000
184430	30JUN2017	\$35,000

Changing how  
values appear  
makes it easier to  
interpret them.



# Formatting Values in Data and Results

```
DATA output-table;  
  SET input-table;  
  FORMAT col-name(s) format;  
RUN;
```

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

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

indicates a character format

total width of the formatted value

All formats include a period.

the number of decimal places for numeric formats

# Common Formats for Numeric Values

Format Name	Example Value	Format Applied	Formatted Value
w. <i>d</i>	12345.67	5.	12346
w. <i>d</i>	12345.67	8.1	12345.7
COMMAw. <i>d</i>	12345.67	COMMA8.1	12,345.7
DOLLARw. <i>d</i>	12345.67	DOLLAR10.2	\$12,345.67
DOLLARw. <i>d</i>	12345.67	DOLLAR10.	\$12,346
YENw. <i>d</i>	12345.67	YEN7.	¥12,346
EUROXw. <i>d</i>	12345.67	EUROX10.2	€12.345,67

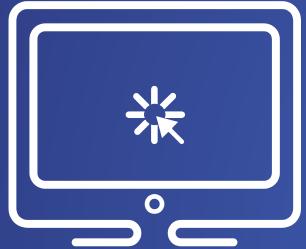
# SAS Format Widths

If you do not specify a format width that is large enough to accommodate a numeric value, the displayed value is automatically adjusted to fit into the width. Numeric values are rounded.

Format	Stored Value	Displayed Value
DOLLAR12.2	27134.2864	\$27,134.29
DOLLAR9.2	27134.2864	\$27134.29
DOLLAR8.2	27134.2864	27134.29
DOLLAR5.2	27134.2864	27134
DOLLAR4.2	27134.2864	27E3

# Common Formats for Date Values

Value	Format	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	WEEKDATE31.	Monday, January 15, 2018
21199	WORDDATE19.	January 15, 2018



# Using Formats

This demonstration illustrates the effects of different width values on date formats.

# Converting Numeric Values to Character Values

numeric to character

Stock	Date
ABC Company	21154
ABC Company	21124
ABC Company	21094
ABC Company	21063

Stock	Date	Day
ABC Company	21154	Fri
ABC Company	21124	Wed
ABC Company	21094	Mon
ABC Company	21063	Fri

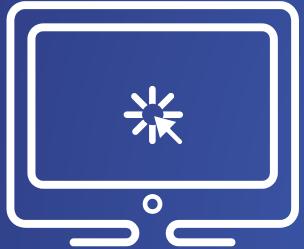
```
Day=put(Date,downname3.);
```

source

format

# SAS Lesson 08

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.



# Explicit Conversion

This demonstration illustrates the explicit conversion of values from numeric to character.

# Converting a Variable to Another Data Type

```
proc contents data=hrdata;  
run;
```

Partial PROC CONTENTS Output

## Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Bonus	Num	8
2	EmpID	Num	8
1	GrossPay	Char	6
4	HireDate	Num	8

How can you convert **GrossPay** to a numeric variable with the same name?

# Quiz

Will this statement convert **GrossPay** to numeric?

```
GrossPay=input(GrossPay,comma6.);
```

No once data is  
read into fd\ cannot  
change long or type  
of variable.



# Explicit Conversion

This demonstration illustrates an attempt to change the variable type with a PUT function.

# Quiz – Correct Answer

Will this statement convert **GrossPay** to numeric?

```
GrossPay=input(GrossPay,comma6.) ;
```

No, **GrossPay** remained a character variable. Why?

# Converting a Variable to Another Data Type

```
GrossPay=input(GrossPay,comma6.);
```



This assignment statement does **not** change **GrossPay** from a character variable to a numeric variable.

A variable is character or numeric. After the variable's type is established, it cannot be changed.

By following three steps, you can create a new variable with the same name and a different type.

# Converting a Variable to Another Data Type

Step 1: Use the RENAME= data set option to rename the variable that you want to convert.

```
data hrdata;  
  set orion.convert(rename=(GrossPay=  
                           CharGross)) ;  
run;
```

General form of the RENAME data set option:

*SAS-data-set(RENAME=(old-name=new-name))*

# Converting a Variable to Another Data Type

Step 2: Use the INPUT function in an assignment statement to create a new variable with the original name of the variable that you renamed.

```
data hrdata;
  set orion.convert(rename=(GrossPay=
                           CharGross));
  GrossPay=input(CharGross,comma6.);
run;
```

# Converting a Variable to Another Data Type

Step 3: Use a DROP= data set option in the DATA statement to exclude the original variable from the output SAS data set.

```
data hrdata;
  set orion.convert(rename=(GrossPay=
                                CharGross));
  drop CharGross;
  GrossPay=input(CharGross,comma6.);
run;
```

The compilation for this program shows the PDV being created with a numeric **GrossPay** variable.

# Converting a Variable: Compilation

```
data hrdata;  
  set orion.convert(rename=(GrossPay=  
                           CharGross));  
  GrossPay=input(CharGross,comma6.);  
  drop CharGross;  
run;
```

## Partial PDV

ID	CharGross	Hired
\$ 5	\$ 6	\$ 7

# Converting a Variable: Compilation

```
data hrdata;  
  set orion.convert(rename=(GrossPay=  
                           CharGross)) ;  
  GrossPay=input(CharGross,comma6.) ;  
  drop CharGross;  
run;
```

## Partial PDV

ID	CharGross	Hired	GrossPay
\$ 5	\$ 6	\$ 7	N 8

# Converting a Variable: Compilation

```
data hrdata;  
  set orion.convert(rename=(GrossPay=  
                           CharGross)) ;  
  GrossPay=input(CharGross,comma6.) ;  
  drop CharGross;  
run;
```

## Partial PDV

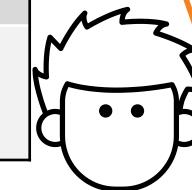
ID \$ 5	CharGross D	Hired \$ 7	GrossPay N 8

## Date Functions

Sas Date represents # of days since 1/1/1960

Function	What It Does
MONTH (SAS-date)	Returns a number from 1 through 12 that represents the month
YEAR (SAS-date)	Returns the four-digit year
DAY (SAS-date)	Returns a number from 1 through 31 that represents the day of the month
WEEKDAY (SAS-date)	Returns a number from 1 through 7 that represents the day of the week (Sunday=1)
QTR (SAS-date)	Returns a number from 1 through 4 that represents the quarter

These functions extract information from SAS date values.



# Date Functions

Function	What It Does
TODAY() or DATE()	Returns the current date as a numeric SAS date value
MDY ( <i>month, day, year</i> )	Returns a SAS date value from month, day, and year values
YRdif ( <i>startdate, enddate, 'AGE'</i> )	Calculates a precise difference in years between two dates
DATDIF( <i>startdate, enddate, basis</i> )	Returns number of days between two dates using selected count convention

# Extracting Data from a Datetime Value

DATEPART(*datetime-value*)

TIMEPART(*datetime-value*)

```
data storm_detail2;
  set pg2.storm_detail;
  WindDate=datepart(ISO_Time);
  WindTime=timepart(ISO_Time);
  format WindDate date9. WindTime time. ;
run;
```

## PDV

ISO_Time	WindDate	WindTime
628192800	7270	21600

Name	ISO_time	WindDate	WindTime
ALBINE	27NOV1979:06:00:00.00	27NOV1979	6:00:00
ALBINE	27NOV1979:12:00:00.00	27NOV1979	12:00:00
ALBINE	27NOV1979:18:00:00.00	27NOV1979	18:00:00
ALBINE	28NOV1979:00:00:00.00	28NOV1979	0:00:00
ALBINE	28NOV1979:06:00:00.00	28NOV1979	6:00:00

# Calculating Date Intervals

```
INTCK('interval', start-date, end-date <, 'method'>)
```

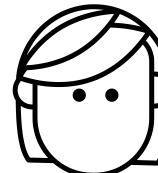
interval that you want to count

SAS date columns

method for calculating intervals

Possible intervals include week, month, year, weekday, or hour.

The INTCK function counts the number of date or time intervals between two events.



# Calculating Date Intervals

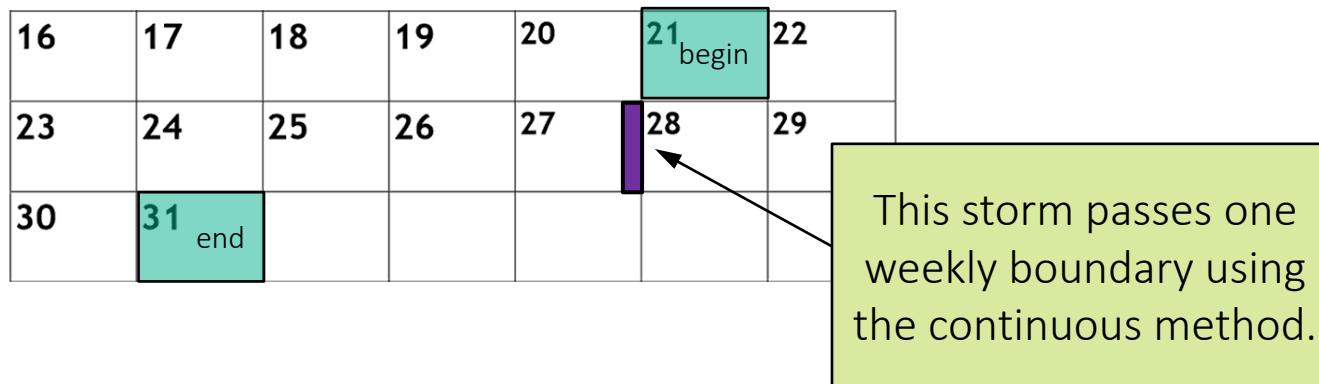
Method	
'discrete' 'd'	Each interval has a fixed boundary. For example, a week ends after Saturday, or a year ends on December 31.



This storm passes two weekly boundaries using the default discrete method.

# Calculating Date Intervals

Method	
'continuous' 'c'	Each interval is measured relative to the start date or time.



# Question

What value would be assigned to **Months2Pay** for each expression?

ServiceDate	PayDate	Months2Pay
10JUL2018	05SEP2018	?

```
Months2Pay=intck ('month' , ServiceDate, PayDate) ;
```

```
Months2Pay=intck ('month' , ServiceDate, PayDate, 'c') ;
```

# Question – Correct Answer

What value would be assigned to **Months2Pay** for each expression?

ServiceDate	PayDate	Months2Pay
10JUL2018	05SEP2018	?

```
Months2Pay=intck ('month' , ServiceDate , PayDate) ;
```

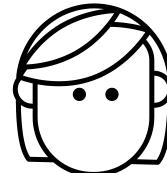
Two end-of-month boundaries were crossed at the end of July and August.

```
Months2Pay=intck ('month' , ServiceDate , PayDate , 'c') ;
```

One month boundary was crossed at August 10. The next boundary will not occur until September 10.

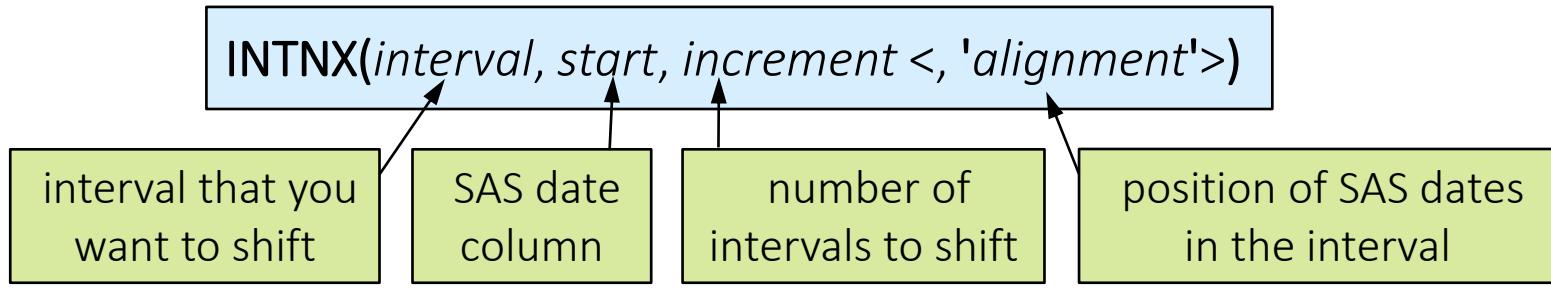
# Shifting Date Values

Customer ID	SalesDate	BillingDate
12808	10JUL2018	01AUG2018
59601	17JUL2018	01AUG2018
42616	02AUG2018	01SEP2018



Suppose you want  
to shift dates to  
the first day of the  
following month.

# Shifting Date Values



## Alignments

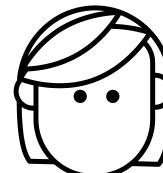
beginning (default)

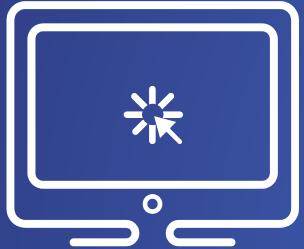
middle

end

same

The `INTNX` function shifts dates or times based on an interval.





# Working with Date Values

This demonstration illustrates using the date functions to create values from existing SAS date columns.

# Business Scenario – Create a List of Charities

A manager in the Finance department asked for a list of all the charities that Orion Star contributes to. She would like to see the name of the charity as well as the ID code assigned to it.

Here is a sketch of the desired output:

Charity Names and ID Codes	
ID	Name
AQI	Aquamissions International
CCI	Cancer Cures, Inc.
CNI	Conserve Nature, Inc.

# Input Data

The **orion.biz\_list** data set is extracted from the accounting system and contains the names of Orion Star's U.S. suppliers, charities, and consultants.

## Partial Listing of **orion.biz\_list**

Acct_	Code	Name
	AEK3	ANGELA E. KEARNEY
	AQI2	AQUAMISSSIONS INTERNATIONAL
	ATS1	A TEAM SPORTS
	CB03	CLAIRE B. OWENS
	CCI2	CANCER CURES, INC.
	CNI2	CONSERVE NATURE, INC.
	CS1	CAROLINA SPORTS

# Input Data – Details

**Acct\_Code** is a character variable defined as length 6.  
Its last digit represents the type of organization: **1** denotes a supplier, **2** a charity, and **3** a consultant.

The other characters in the **Acct\_Code** variable represent the ID for the organization, so the **ID** value can have as many as five characters.

Example:

Acct_Code	ID
\$6	\$5
AQI2	AQI

- 2 denotes a charity.
- AQI is the ID.

# Input Data – Details

The name of the organization is stored as all capital letters. In the desired output, only the first letter of each word is capitalized.

Example:

Name
AQUAMISSSIONS INTERNATIONAL

Change to:

Name
Aquamissions International

# Business Scenario – Desired Results

Create a new data set, **charities**, that has the information that the finance manager would like to see.

Partial Listing of **charities**

ID	Acct_Code	Name
AQI	AQI2	Aquamissions International
CCI	CCI2	Cancer Cures, Inc.
CNI	CNI2	Conserve Nature, Inc.
CS	CS2	Child Survivors
CU	CU2	Cuidadores Ltd.
DAI	DAI2	Disaster Assist, Inc.

This data set can then be used to create the manager's report.

# Create the List of Charities – Step 1

The first step is to subset the data based on the last character of **Acct\_Code**.

Partial Listing of **orion.biz** list

Acct_Code	Name
AEK3	ANGELA E. KEARNEY
AQI2	AQUAMISSESS INTERNATIONAL
ATS1	A TEAM SPORTS
CB03	CLAIRE B. OWENS
CCI2	CANCER CURES, INC.
CNI2	CONSERVE NATURE, INC.
CS1	CAROLINA SPORTS

Charities

SAS character functions make this task easy.

# The SUBSTR Function (Right Side) \*

The SUBSTR function on the right side of an assignment statement is used to extract characters.

General form of the SUBSTR function:

NewVar=SUBSTR(*string,start<,length>*);

<i>string</i>	can be a character constant, variable, or expression.
<i>start</i>	specifies the starting position.
<i>length</i>	specifies the number of characters to extract. If omitted, the substring consists of the remainder of string.
<i>NewVar</i>	If <i>NewVar</i> is a new variable it will be created with the same length as <i>string</i> . To set a different length for <i>NewVar</i> , use a LENGTH statement prior to the assignment statement.

# The SUBSTR Function – Example

Extract the first three characters from the value in the **Item\_Code** variable and store them in **Item\_Type**.

```
Item_Type=substr(Item_Code,1,3);
```

PDV

Item_Code	Item_Type
\$ 20	\$ 20
978-1-59994-397-8	978

Starting at position 1  
for a length of 3

# Setup for the Poll

This is the current value of **Item\_Code**:

PDV
Item_Code
\$ 20
978-1-59994-397-8

The SUBSTR function is a good method to extract the highlighted digits.

# Multiple Choice Poll

Which SUBSTR function can extract the group of five numbers from the middle of the **Item\_Code** value?

- a. `substr(Item_Code, 5, 7)`
- b. `substr(Item_Code, 5)`
- c. `substr(Item_Code, 7, 5)`
- d. `substr(Item_Code, 'mid', 5)`



PDV
Item_Code
\$ 20
978-1-59994-397-8

# Multiple Choice Poll – Correct Answer

Which SUBSTR function can extract the group of five numbers from the middle of the **Item\_Code** value?

- a. `substr(Item_Code, 5, 7)`
- b. `substr(Item_Code, 5)`
- c. `substr(Item_Code, 7, 5)`
- d. `substr(Item_Code, 'mid', 5)`

# Create the List of Charities – Step 1

The last non-blank character in the **Acct\_Code** value occurs in different positions for different observations.

Partial Listing of  
**orion.biz\_list**

Acct_Code	Last character in position 4
AEK3	
AQI2	
ATS1	Last character in position 3
CB03	
CCI2	
CNI2	
CS1	
CS2	
CU2	

You need some way to determine the position of the last character so that the SUBSTR function can extract it.

# The LENGTH Function

The LENGTH function returns the length of a non-blank character string, excluding trailing blanks.



General form of the LENGTH function:

*NewVar=LENGTH(argument);*

Example:  
`Code = 'ABCD ';`  
`Last_NonBlank=length (Code) ;`

## PDV

Code	Last_NonBlank
\$ 6	N 8
ABCD	4

# Create the List of Charities – Step 1

This program uses the SUBSTR and LENGTH functions to create the **charities** data set.

The LENGTH function is *nested*, or used as an argument to the SUBSTR function.

```
data charities;
  length ID $ 5;
  set orion.biz_list;
  if substr(Acct_Code, length(Acct_Code), 1)='2';
  ID=substr(Acct_Code, 1, length(Acct_Code)-1);
run;
```

Partially stepping through the execution for the first charity observation shows how the functions transform the data.

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Read the first  
charity observation.

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

4

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

PDV

ID \$ 5	Acct Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

True

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

3

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

## PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
AQI	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Implicit OUTPUT;  
Implicit RETURN;

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
AQI	AQI2	AQUAMISSESS INTERNATIONAL

# Create the List of Charities – Step 1 Complete

## Listing of **charities**

ID	Acct_Code	Name
AQI	AQI2	AQUAMISSESS INTERNATIONAL
CCI	CCI2	CANCER CURES, INC.
CNI	CNI2	CONSERVE NATURE, INC.
CS	CS2	CHILD SURVIVORS
CU	CU2	CUIDADORES LTD.
DAI	DAI2	DISASTER ASSIST, INC.
ES	ES2	EARTHSALVORS
FFC	FFC2	FARMING FOR COMMUNITIES
MI	MI2	MITLEID INTERNATIONAL
SBA	SBA2	SAVE THE BABY ANIMALS
V2	V22	VOX VICTIMAS
YYCR	YYCR2	YES, YOU CAN RECYCLE

Step 2 is to transform the values in **Name** to a mix of uppercase and lowercase.

# The PROPCASE Function

The PROPCASE function converts all words in an argument to *proper case*, in which the first letter is uppercase and the remaining letters are lowercase.

General form for the PROPCASE function:

```
NewVar=PROPCASE(argument <,delimiter(s)>);
```

<i>argument</i>	can be a character constant, variable, or expression.
<i>delimiter(s)</i>	delimiters are characters which separate words. If omitted, the default delimiters are the blank, /, -, (, , ., and tab characters.
<i>NewVar</i>	If <i>NewVar</i> is a new variable, it is created with the same length as <i>argument</i> .

# The PROPCASE Function

Example:

```
Name = 'SURF&LINK SPORTS' ;  
Pname = propcase(Name) ;  
Pname2 = propcase(Name, ' &' ) ;
```

## PDV

Name	Pname
\$ 16	\$ 16
SURF&LINK SPORTS	Surf&link Sports

Pname2
\$ 16
Surf&Link Sports

# Quiz

This PDV shows the current value of **Name**:

Name
HEATH*BARR*LITTLE EQUIPMENT SALES

Write an assignment statement that converts  
the value of **Name** to this:

Name
Heath*Barr*Little Equipment Sales

# Quiz – Correct Answer

This PDV shows the current value of **Name**:

Name
HEATH*BARR*LITTLE EQUIPMENT SALES

Write an assignment statement that will convert the value of **Name** to this:

Name
Heath*Barr*Little Equipment Sales

```
Name = propcase(Name, ' * ') ;
```

The second argument to the PROPCASE function must list all the characters to use as delimiters. In this example, the space and \* both need to be listed.

## Create the List of Charities – Step 2

Adding an assignment statement to convert **Name** to proper case completes the **charities** data set.

```
data charities;
  length ID $ 5;
  set orion.biz_list;
  if substr(Acct_Code,length(Acct_Code),1)='2';
  ID=substr(Acct_Code,1,length(Acct_Code)-1);
  Name = propcase(Name);
run;
```

# Create the List of Charities – Complete

## Listing of **charities**

ID	Acct_Code	Name
AQI	AQI2	Aquamissions International
CCI	CCI2	Cancer Cures, Inc.
CNI	CNI2	Conserve Nature, Inc.
CS	CS2	Child Survivors
CU	CU2	Cuidadores Ltd.
DAI	DAI2	Disaster Assist, Inc.
ES	ES2	Earthsaviors
FFC	FFC2	Farming For Communities
MI	MI2	Mitleid International
SBA	SBA2	Save The Baby Animals
V2	V22	Vox Victimas
YYCR	YYCR2	Yes, You Can Recycle

# Other Useful Character Functions

Function	Purpose
RIGHT( <i>string</i> )	right-aligns a character expression.
LEFT( <i>string</i> )	left-aligns a character expression.
UPCASE( <i>string</i> )	converts all letters in an argument to uppercase.
LOWCASE( <i>string</i> )	converts all letters in an argument to lowercase.
CHAR( <i>string,position</i> )	returns a single character from a specified <i>position</i> in a character <i>string</i> .

# Quiz

Find the syntax error in the code below. Product\_Name has a length of 45.

Partial listing of product\_list:

Product_ID	Product_Name	Supplier_ID	Product_Level	Product_Ref_ID
220100700023	Armadillo Road Dmx Men's Running Shoes	16733	1	220100700000
220100700024	Armadillo Road Dmx Women's Running Shoes	16733	1	220100700000
220100700046	Tcp 6 Men's Running Shoes	16733	1	220100700000

```
data shoes;
  set orion.product_list;
  if substr(right(Product_Name,33,13))=
    'Running Shoes';
run;
```

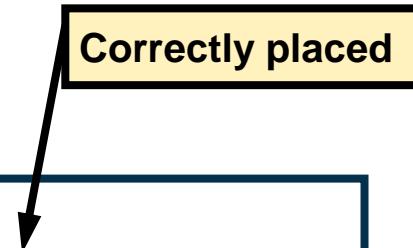
# Quiz – Correct Answer

Misplaced parentheses are some of the most common syntax errors with functions.

Corrected program:

```
data shoes;
  set orion.product_list;
  if substr(right(Product_Name),33,13)=
    'Running Shoes';
run;
```

Correctly placed



# SAS Lesson 09

Portions Copyright © 2018 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

# Input Data

The **orion.biz\_list** data set is extracted from the accounting system and contains the names of Orion Star's U.S. suppliers, charities, and consultants.

## Partial Listing of **orion.biz\_list**

Acct_	Code	Name
	AEK3	ANGELA E. KEARNEY
	AQI2	AQUAMISSSIONS INTERNATIONAL
	ATS1	A TEAM SPORTS
	CB03	CLAIRE B. OWENS
	CCI2	CANCER CURES, INC.
	CNI2	CONSERVE NATURE, INC.
	CS1	CAROLINA SPORTS

# Input Data – Details

**Acct\_Code** is a character variable defined as length 6. Its last digit represents the type of organization: **1** denotes a supplier, **2** a charity, and **3** a consultant.

The other characters in the **Acct\_Code** variable represent the ID for the organization, so the **ID** value can have as many as five characters.

Example:

Acct_Code	ID
\$6	\$5
AQI2	AQI

- 2 denotes a charity.
- AQI is the ID.

# Create the List of Charities – Step 1

This program uses the SUBSTR and LENGTH functions to create the **charities** data set.

The LENGTH function is *nested*, or used as an argument to the SUBSTR function.

```
data charities;
  length ID $ 5;
  set orion.biz_list;
  if substr(Acct_Code, length(Acct_Code), 1)='2';
  ID=substr(Acct_Code, 1, length(Acct_Code)-1);
run;
```

Partially stepping through the execution for the first charity observation shows how the functions transform the data.

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Read the first  
charity observation.

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

4

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

PDV

ID \$ 5	Acct Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

True

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

3

## PDV

ID	Acct_Code	Name
\$ 5	\$ 6	\$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

## PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
AQI	AQI2	AQUAMISSESS INTERNATIONAL

# Execution: Step 1

```
data charities;
length ID $ 5;
set orion.biz_list;
if substr(Acct_Code,length(Acct_Code),1)='2';
ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Implicit OUTPUT;  
Implicit RETURN;

PDV

ID \$ 5	Acct_Code \$ 6	Name \$ 30
AQI	AQI2	AQUAMISSESS INTERNATIONAL

# Create the List of Charities – Step 1 Complete

## Listing of **charities**

ID	Acct_Code	Name
AQI	AQI2	AQUAMISSESS INTERNATIONAL
CCI	CCI2	CANCER CURES, INC.
CNI	CNI2	CONSERVE NATURE, INC.
CS	CS2	CHILD SURVIVORS
CU	CU2	CUIDADORES LTD.
DAI	DAI2	DISASTER ASSIST, INC.
ES	ES2	EARTHSALVORS
FFC	FFC2	FARMING FOR COMMUNITIES
MI	MI2	MITLEID INTERNATIONAL
SBA	SBA2	SAVE THE BABY ANIMALS
V2	V22	VOX VICTIMAS
YYCR	YYCR2	YES, YOU CAN RECYCLE

Step 2 is to transform the values in **Name** to a mix of uppercase and lowercase.

# The PROPCASE Function

AF

The PROPCASE function converts all words in an argument to *proper case*, in which the first letter is uppercase and the remaining letters are lowercase.

General form for the PROPCASE function:

```
NewVar=PROPCASE(argument <,delimiter(s)>);
```

<i>argument</i>	can be a character constant, variable, or expression.
<i>delimiter(s)</i>	delimiters are characters which separate words. If omitted, the default delimiters are the blank, /, - , ( , ., and tab characters.
<i>NewVar</i>	If <i>NewVar</i> is a new variable, it is created with the same length as <i>argument</i> .

# The PROPCASE Function

Example:

```
Name = 'SURF&LINK SPORTS' ;
Pname = propcase(Name) ;
Pname2 = propcase(Name, ' &' ) ;
```

## PDV

Name	Pname
\$ 16	\$ 16
SURF&LINK SPORTS	Surf&link Sports

Pname2
\$ 16
Surf&Link Sports

# Quiz

This PDV shows the current value of **Name**:

Name
HEATH*BARR*LITTLE EQUIPMENT SALES

Write an assignment statement that converts  
the value of **Name** to this:

Name
Heath*Barr*Little Equipment Sales

# Quiz – Correct Answer

This PDV shows the current value of **Name**:

Name
HEATH*BARR*LITTLE EQUIPMENT SALES

Write an assignment statement that will convert the value of **Name** to this:

Name
Heath*Barr*Little Equipment Sales

```
Name = propcase(Name, ' * ') ;
```

The second argument to the PROPCASE function must list all the characters to use as delimiters. In this example, the space and \* both need to be listed.

## Create the List of Charities – Step 2

Adding an assignment statement to convert **Name** to proper case completes the **charities** data set.

```
data charities;
  length ID $ 5;
  set orion.biz_list;
  if substr(Acct_Code,length(Acct_Code),1)='2';
  ID=substr(Acct_Code,1,length(Acct_Code)-1);
  Name = propcase(Name);
run;
```

# Create the List of Charities – Complete

## Listing of **charities**

ID	Acct_Code	Name
AQI	AQI2	Aquamissions International
CCI	CCI2	Cancer Cures, Inc.
CNI	CNI2	Conserve Nature, Inc.
CS	CS2	Child Survivors
CU	CU2	Cuidadores Ltd.
DAI	DAI2	Disaster Assist, Inc.
ES	ES2	Earthsaviors
FFC	FFC2	Farming For Communities
MI	MI2	Mitleid International
SBA	SBA2	Save The Baby Animals
V2	V22	Vox Victimas
YYCR	YYCR2	Yes, You Can Recycle

# Other Useful Character Functions

Function	Purpose
RIGHT( <i>string</i> )	right-aligns a character expression.
LEFT( <i>string</i> )	left-aligns a character expression.
UPCASE( <i>string</i> )	converts all letters in an argument to uppercase.
LOWCASE( <i>string</i> )	converts all letters in an argument to lowercase.
CHAR( <i>string,position</i> )	returns a single character from a specified <i>position</i> in a character <i>string</i> .

subset out of any 1 character -

↑ Spacing  
padding  
of strings.  
Words.

# Quiz

Find the syntax error in the code below. Product\_Name has a length of 45.

Partial listing of product\_list:

Product_ID	Product_Name	Supplier_ID	Product_Level	Product_Ref_ID
220100700023	Armadillo Road Dmx Men's Running Shoes	16733	1	220100700000
220100700024	Armadillo Road Dmx Women's Running Shoes	16733	1	220100700000
220100700046	Tcp 6 Men's Running Shoes	16733	1	220100700000

```
data shoes;
  set orion.product_list;
  if substr(right(Product_Name,33,13))=
    'Running Shoes';
run;
```

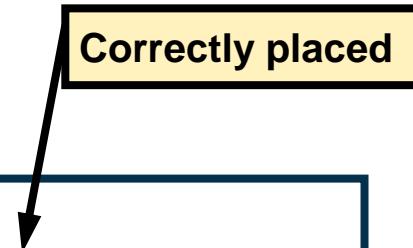
# Quiz – Correct Answer

Misplaced parentheses are some of the most common syntax errors with functions.

Corrected program:

```
data shoes;
  set orion.product_list;
  if substr(right(Product_Name),33,13)=
    'Running Shoes';
run;
```

Correctly placed



# Business Scenario – Create Mailing List Data

The **orion.contacts** data set contains the contact information for each charity's representative.

Partial Listing of **orion.contacts**

ID	Title	Name	Address1	Address2
AQI	Ms.	Farr, Sue	15 Harvey Rd.	Macon, GA 31298
CCI	Dr.	Cox, Kay B.	163 McNeil Pl.	Kern, CA 93280
CNI	Mr.	Mason, Ron	442 Glen Ave.	Miami, FL 33054
CS	Ms.	Ruth, G. H.	2491 Brady St.	Munger, MI 48747

**Address1** and **Address2** are in the correct form to use for a mailing address, but the **Title** and **Name** variables need to be combined into a new variable, **FullName**.

# Business Scenario – Desired Output

Create a new data set, **labels**, that is suitable for creating mailing labels.

Partial Listing of **labels**

ID	FullName	Address1	Address2
AQI	Ms. Sue Farr	15 Harvey Rd.	Macon, GA 31298
CCI	Dr. Kay B. Cox	163 McNeil Pl.	Kern, CA 93280
CNI	Mr. Ron Mason	442 Glen Ave.	Miami, FL 33054
CS	Ms. G. H. Ruth	2491 Brady St.	Munger, MI 48747

# Create Mailing List Data

Partial Listing of `orion.contacts`

Title	Name
Ms.	Farr, Sue
Dr.	Cox, Kay B.
Mr.	Mason, Ron
Ms.	Ruth, G. H.
Prof.	Florentino, Helen-Ashe H
Ms.	Van Allsburg, Jan F.
Mr.	Laff, Stanley X.
Mr.	Rizen, George Q.
Dr.	Mitchell, Marc J.
Ms.	Mills, Dorothy E.
Dr.	Webb, Jonathan W.
Mr.	Keenan, Maynard J.

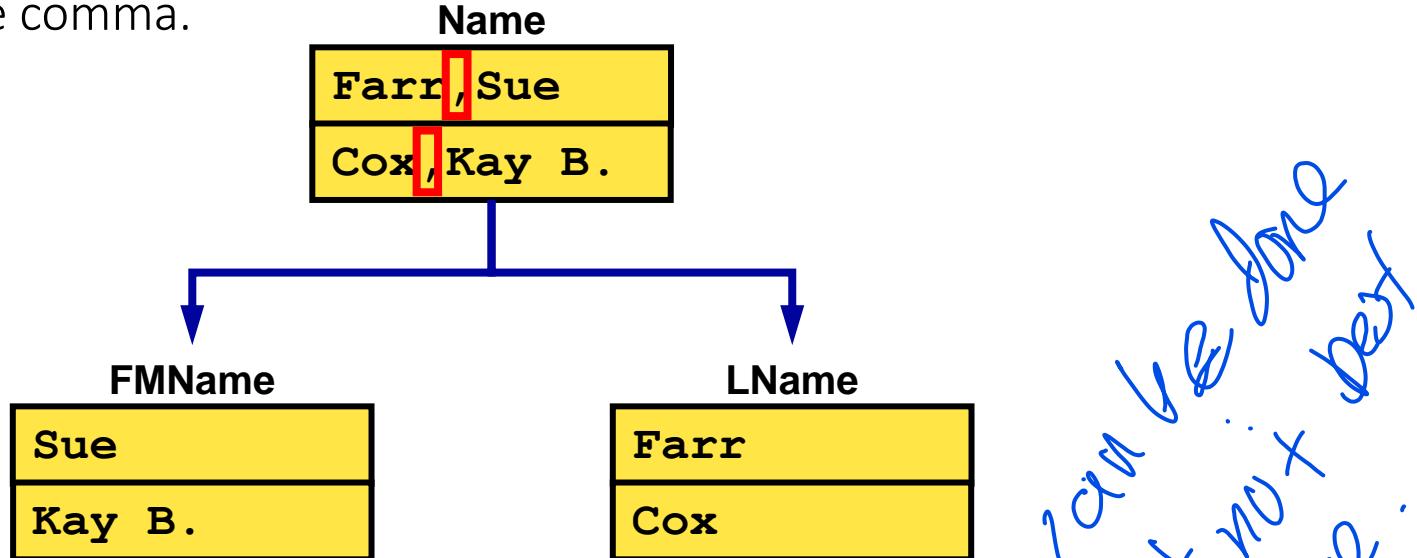
Two steps need to be accomplished:

**Step 1:** Separate the last name from the first and middle names.

**Step 2:** Combine the title, the first and middle names, and the last name.

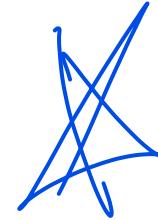
# Separating Data Elements

The first step in creating the mailing list is to separate the contact's name into two parts based on the position of the comma.



Would the SUBSTR function be appropriate for this?

# The SCAN Function



The SCAN function returns the *n*th word of a character value.

General form of the SCAN function:

**NewVar=SCAN(*string,n<,charlist>*);**

<i>string</i>	can be a character constant, variable, or expression.
<i>n</i>	specifies the <i>n</i> th word to extract from <i>string</i> .
<i>charlist</i>	lists the character(s) that delimit words. If omitted, the default delimiters are as follows:
ASCII (PC, UNIX)	blank . < ( +   & ! \$ * ) ; - / , % ^
EBCDIC (z/OS)	blank . < ( +   & ! \$ * ) ; - / , %   ¢ ¬



## The SCAN Function – Details

When you use the SCAN function,

- a missing value is returned if there are fewer than  $n$  words in the string
- if  $n$  is negative, the SCAN function selects the word in the character string starting from the end of string
- the length of a new created variable is the length of the first argument.



A good practice is to explicitly define the length of any created variable with a LENGTH statement.

# The SCAN Function – Details

When you use the SCAN function,

- delimiters before the first word have no effect
- any character or set of characters can serve as delimiters
- two or more contiguous delimiters are treated as a single delimiter
- modifiers can be used as a fourth argument to change the default behavior.

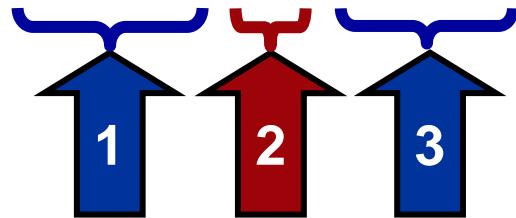
# The SCAN Function – Example

Extract the second word of **Phrase**.

```
Second=scan(Phrase,2,' ');
```

PDV

Phrase	Second
\$ 21	\$ 21
software and services	and



# Quiz

Consider this PDV and assignment statement:

```
Second=scan(Phrase,2,',');
```

**PDV**

Phrase	Second
\$ 28	\$ 28
software, hardware, services	

What value will be stored in **Second**?

# Quiz – Correct Answer

Consider this PDV and assignment statement:

```
Second=scan(Phrase,2,' ','');
```

PDV

Phrase	Second
\$ 28	\$ 28
software, hardware, services	hardware



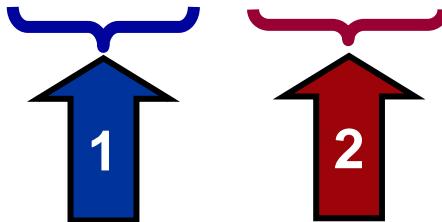
# The SCAN Function – Example

Extract the second word of **Phrase** without the leading space.

```
Second=scan(Phrase,2,' ','');
```

PDV

Phrase	Second
\$ 28	\$ 28
software, hardware, services	hardware



# Multiple Choice Poll

What expression completes the assignment statement to correctly extract 2007 from the **Text** variable?

- a. **scan(Text, -1);**
- b. **scan(Text, 6);**
- c. **scan(Text, 6, ', ');**
- d. All of the above would work.

```
data Scan_Quiz;  
  Text = "New Year's Day, January 1st, 2007";  
  Year = ?;  
run;
```

# Multiple Choice Poll – Correct Answer

What expression completes the assignment statement to correctly extract 2007 from the **Text** variable?

- a. `scan(Text, -1) ;`
- b. `scan(Text, 6) ;`
- c. `scan(Text, 6, ', ') ;`
- d. All of the above would work.

# Create Mailing List Data

Using the SCAN function gives an easy way to separate the names for the mailing list.

```
data labels;  
  set orion.contacts;  
  length FMName LName $ 15;  
  FMName = scan(Name,2,',');  
  LName = scan(Name,1,',');  
run;
```

# Create Mailing List Data

```
proc print data=labels noobs;  
    var ID Name Title FMName LName;  
run;
```

Partial PROC PRINT Output

ID	Name	Title	FMName	LName
AQI	Farr, Sue	Ms.	Sue	Farr
CCI	Cox, Kay B.	Dr.	Kay B.	Cox
CNI	Mason, Ron	Mr.	Ron	Mason
CS	Ruth, G. H.	Ms.	G. H.	Ruth

The next step is to join the values of **Title**, **FMName**, and **LName** into another variable.

# The CATX Function

The CATX function joins or *concatenates* character strings.

General form of the CATX function:

NewVar = CATX(separator, string-1, ... ,string-n)

separator Is a character string that is inserted between the concatenated *string-1,...,string-n* arguments.

*string-1,*  
*...,string-n* can be a character constant, variable, or expression.  
Leading and trailing blanks are removed from each argument.

The size of the created variable, *NewVar*, is 200 bytes if it is not previously defined with a LENGTH statement.

→ leading and trailing  
blanks  
are  
removed.

# The CATX Function – Example

Combine **FMName** and **LName** to create **FullName**.

```
FullName=catx(' ',FMName,LName);
```

PDV

FMName	LName	FullName
\$ 15	\$ 15	\$ 200
Sue	Farr	Sue Farr

# Other CAT Functions

There are three other CAT functions that concatenate character strings.

Function	Details
CAT( <i>string-1</i> , ..., <i>string-n</i> )	does not remove leading or trailing blanks from the arguments before concatenating them.
CATS( <i>string-1</i> , ..., <i>string-n</i> )	removes leading and trailing blanks from the arguments.
CATT( <i>string-1</i> , ..., <i>string-n</i> )	removes trailing blanks from the arguments.

CATX removes leading & trailing blanks.

# Create Mailing List Data – Finished Program

Adding an assignment statement with the CATX function completes the program.

```
data labels;
  set orion.contacts;
  length FullName $ 35 FMName LName $ 15;
  FMName = scan(Name,2,',','');
  LName = scan(Name,1,',','');
  FullName = catx(' ',Title,FMName,LName);
run;
```

# Create Mailing List Data – Finished Program

```
proc print data=labels noobs;  
    var ID FullName Address1 Address2;  
run;
```

Partial PROC PRINT Output

ID	FullName	Address1	Address2
AQI	Ms. Sue Farr	15 Harvey Rd.	Macon, GA 31298
CCI	Dr. Kay B. Cox	163 McNeil Pl.	Kern, CA 93280
CNI	Mr. Ron Mason	442 Glen Ave.	Miami, FL 33054
CS	Ms. G. H. Ruth	2491 Brady St.	Munger, MI 48747

# Concatenation Operator

The *concatenation operator* is another way to join character strings.  
General form of the concatenation operator:

```
NewVar=string1 !! string2;
```

Example: **Phone = '(! ! ! area ! ! !) ! ! ! Number';**

## PDV

Area	Number	Phone
\$ 3	\$ 8	\$ 14
919	531-0000	(919) 531-0000

-  The operator can also be written as two vertical bars (||) or two broken vertical bars (|||).

# Business Scenario: Data Clean Up

The Internet Sales Group accidentally used the wrong data files for the Orion Star Catalog Web site. They corrected the problem as soon as it was noticed, but some orders were created with data errors in them.

**orion.clean\_up** has sample observations showing the problems.

# Business Scenario: Data Clean Up

Listing of `orion.clean_up`

Product_ID	Product	Order_ID
21 02 002 00003	Sunfit Trunks, Blue	1231986335
21 02 002 00003	Luci Knit Mittens, Red	1232003930
21 02 002 00004	Luci Knit mittens, Blue	1232007693
21 02 002 00004	Sunfit Trunks, aqua	1232007700
21 02 002 00005	Sunfit Trunks, Yellow	1232087464
21 02 002 00005	Lucky Knit Mittens, Black	1232092527

- The **Product\_ID** for mittens should have 5 instead of a 2 for the third group of numbers.
- Luci is a typo; the correct word is Lucky.
- **Product\_ID** values should have no internal spaces.
- All words in the **Product** value should start with a capital letter.

# Business Scenario – Desired Output

The **correct** data set shows what the data should be.

## Listing of **correct**

Product_ID	Product	Order_ID
210200200003	Sunfit Trunks, Blue	1231986335
210200500003	Lucky Knit Mittens, Red	1232003930
210200500004	Lucky Knit Mittens, Blue	1232007693
210200200004	Sunfit Trunks, Aqua	1232007700
210200200005	Sunfit Trunks, Yellow	1232087464
210200500005	Lucky Knit Mittens, Black	1232092527

# Data Clean Up – Step 1

The first step in creating the **correct** data set is to do the following:

- Find the observations with Mittens as part of the **Product** value.
- Change the middle characters of the **Product\_ID** values for those observations.

The FIND and SUBSTR functions are useful for this.



# The FIND Function

The FIND function searches a target string for a specified substring.

General form of the FIND function:

*Position = FIND(string,substring<,modifiers,startpos>);*

The FIND function returns a numeric value that is

- the starting position of the first occurrence of *substring* within *string*, if *substring* is found
- 0, if *substring* is not found.

# ~~A~~ The FIND Function

The FIND function searches a target *string* for a specified *substring*.

General form of the FIND function:

*Position* = FIND(*string*,*substring*<,*modifiers*,*startpos*>);

*Modifiers* can be

- I to indicate a case-insensitive search
- T to indicate to ignore trailing blanks in the string and substring values.

*startpos* indicates where in the *string* to start searching for the *substring*.

# The INDEX Function

The INDEX function searches a target *string* for a specified *substring*.

INDEX is a predecessor to the FIND function and performs the same operation without the optional arguments.

General form of the INDEX function:

*Position* = INDEX(*string*, *substring*);

# The FIND Function – Example

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

**PDV**

Pos1
N 8

What value will SAS assign to **Pos1**?

# The FIND Function – Example

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

2

**PDV**

Pos1
N 8
2

# The FIND Function – Example

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

20

## PDV

Pos1	Pos2
N 8	N 8
2	20

# Quiz

Complete the PDV for the values for **Pos3** and **Pos4**.

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

## PDV

Pos1	Pos2	Pos3	Pos4
N 8	N 8	N 8	N 8
2	20		

# Quiz – Correct Answer

Complete the PDV for the values for **Pos3** and **Pos4**.

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

## PDV

Pos1	Pos2	Pos3	Pos4
N 8	N 8	N 8	N 8
2	20	0	2

# The FIND Function – Example

```
data find;  
  Text='AUSTRALIA, DENMARK, US';  
  Pos1=find(Text,'US');  
  Pos2=find(Text,' US');  
  Pos3=find(Text,'us');  
  Pos4=find(Text,'us','I');  
  Pos5=find(Text,'us','I',10);  
run;
```

21



## PDV

Pos1	Pos2	Pos3	Pos4	Pos5
N 8	N 8	N 8	N 8	N 8
2	20	0	2	21

# The SUBSTR Function (Left Side)

This form of the SUBSTR function (left side of assignment statement) replaces characters in a character variable.

General form of the SUBSTR function (left side):

**SUBSTR(*string,start<,length>*)=value;**

<i>string</i>	specifies a character variable.
<i>start</i>	specifies the starting position to replace characters with the <i>value</i> .
<i>length</i>	specifies the number of characters to replace in <i>string</i> . If omitted, all characters from the <i>start</i> position to the end of the <i>string</i> are replaced. The length value cannot be larger than the remaining length of <i>string</i> (including trailing blanks) after <i>start</i> .

# The SUBSTR Function (Left Side) – Example

Replace two characters starting at position 11.

11

```
Location = 'Columbus, GA 43227';  
substr(Location,11,2)='OH';
```

**PDV**

Location
\$ 18
Columbus, OH 43227

# Data Clean Up – Step 1

Use the SUBSTR and FIND functions to change incorrect product IDs for mittens.

```
data correct;
  set orion.clean_up;
  if find(Product,'Mittens','I')>0 then do;
    substr(Product_ID,9,1) = '5';
  end;
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 1

PROC PRINT Output

Product_ID	Product	Order_ID
21 02 002 00003	Sunfit Trunks, Blue	1231986335
21 02 005 00003	Luci Knit Mittens, Red	1232003930
21 02 005 00004	Luci Knit mittens, blue	1232007693
21 02 002 00004	Sunfit Trunks, aqua	1232007700
21 02 002 00005	Sunfit Trunks, Yellow	1232087464
21 02 005 00005	Lucky Knit Mittens, Black	1232092527

The next step is to change the error Luci to Lucky.

The TRANWRD function is the best way to do this kind of change.

# The TRANWRD Function

The TRANWRD function replaces or removes all occurrences of a given word (or a pattern of characters) within a character string.

General form for the TRANWRD function:

```
NewVar=TRANWRD(source,target,replacement);
```

<i>source</i>	specifies the source string that you want to change.
<i>target</i>	specifies the string searched for in <i>source</i> .
<i>replacement</i>	specifies the string that replaces <i>target</i> .

# The TRANWRD Function – Details

General form for the TRANWRD function:

*NewVar=TRANWRD(source,target,replacement);*

These details apply when you use the TRANWRD function:

- The TRANWRD function does not remove trailing blanks from *target* or *replacement*.
- If *NewVar* was not previously defined, it is given a length of 200.
- If the target string is not found in the source, then no replacement occurs.

## Data Clean Up – Step 2

Use the TRANWRD function to replace all occurrences of Luci with Lucky.

```
data correct;
  set orion.clean_up;
  if find(Product,'Mittens','I') > 0 then do;
    substr(Product_ID,9,1) = '5';
    Product=Tranwrd(Product,'Luci ','Lucky ');
  end;
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 2

PROC PRINT Output

Product_ID	Product	Order_ID
21 02 002 00003	Sunfit Trunks, Blue	1231986335
21 02 005 00003	Lucky Knit Mittens, Red	1232003930
21 02 005 00004	Lucky Knit mittens, blue	1232007693
21 02 002 00004	Sunfit Trunks, aqua	1232007700
21 02 002 00005	Sunfit Trunks, Yellow	1232087464
21 02 005 00005	Lucky Knit Mittens, Black	1232092527

For step 3, removing the embedded blanks from **Product\_ID** is easy with the COMPRESS function.

# The COMPRESS Function

The COMPRESS function removes the characters listed in the *chars* argument from the *source*.

General form for the COMPRESS function:

```
NewVar=COMPRESS(source<,chars> <,modifiers>);
```

Modifiers control whether specified characters are kept or removed. They can also add specific types of characters to the list as a group. See Prep Guide for full list.

If no optional arguments are specified, the COMPRESS function removes all blanks from the *source*.

# The COMPRESS Function

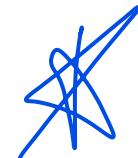
```
ID ='20 01-005 024';
New_ID1=compress(ID);
New_ID2=compress(ID,'-');
New_ID3=compress(ID,' -');
```

## PDV

ID	New_ID1
\$ 13	\$ 13
20 01-005 024	2001-005024

New_ID2	New_ID3
\$ 13	\$ 13
20 01005 024	2001005024

# Other Functions That Remove Blanks



Function	Purpose
TRIM( <i>string</i> )	removes trailing blanks from a character string.
STRIP( <i>string</i> )	removes all leading and trailing blanks from a character string.
COMPBL( <i>string</i> )	removes multiple blanks from a character string by translating each occurrence of two or more consecutive blanks into a single blank.

# Data Clean Up – Step 3

Use the COMPRESS and PROPCASE functions to eliminate blanks from **Product\_ID** and ensure the proper case for **Product**.

```
data correct;
  set orion.clean_up;
  if find(Product,'Mittens','I') > 0 then do;
    substr(Product_ID,9,1) = '5';
    Product=tranwrd(Product,'Luci ','Lucky ');
  end;
  Product_ID = compress(Product_ID);
  Product = propcase(Product);
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 3

PROC PRINT Output

Product_ID	Product	Order_ID
210200200003	Sunfit Trunks, Blue	1231986335
210200500003	Lucky Knit Mittens, Red	1232003930
210200500004	Lucky Knit Mittens, Blue	1232007693
210200200004	Sunfit Trunks, Aqua	1232007700
210200200005	Sunfit Trunks, Yellow	1232087464
210200500005	Lucky Knit Mittens, Black	1232092527

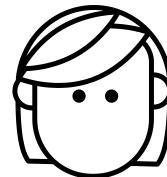
# Manipulating Data with Functions

Using Numeric Functions

# Changing Numeric Precision

Function	What it does
ROUND( <i>number</i> )	Returns a value rounded to nearest multiple
CEIL( <i>number</i> )	Returns the smallest integer that is greater than or equal to the argument
FLOOR( <i>number</i> )	Returns the largest integer that is less than or equal to the argument
INT( <i>number</i> )	Returns the integer value

These functions can be used to truncate decimal values.



# The ROUND Function

The ROUND function returns a value rounded to the nearest multiple of the round-off unit.

General form of the ROUND function:

`NewVar=ROUND(argument<,round-off-unit>);`

*argument* is a number or numeric expression.

*round-off-unit* is numeric and positive. If *round-off-unit* is not provided, *argument* is rounded to the nearest integer.

# The ROUND Function – Example

```
data truncate;  
    NewVar1=round(12.12);  
    NewVar2=round(42.65,.1);  
    NewVar3=round(-6.478);  
    NewVar4=round(96.47,10);  
run;
```

## PDV

NewVar1 N 8	NewVar2 N 8	NewVar3 N 8	NewVar4 N 8
12	42.7	-6	100

# The ROUND Function – Example

```
data truncate;  
    NewVar5=round(12.69,.25);  
    NewVar6=round(42.65,.5);  
run;
```

Round to the nearest multiple of .25

**PDV**

NewVar5	NewVar6
N 8	N 8
12.75	.

# The ROUND Function – Example

```
data truncate;  
    NewVar5=round(12.69,.25);  
    NewVar6=round(42.65,.5);  
run;
```

Round to the nearest multiple of .5

**PDV**

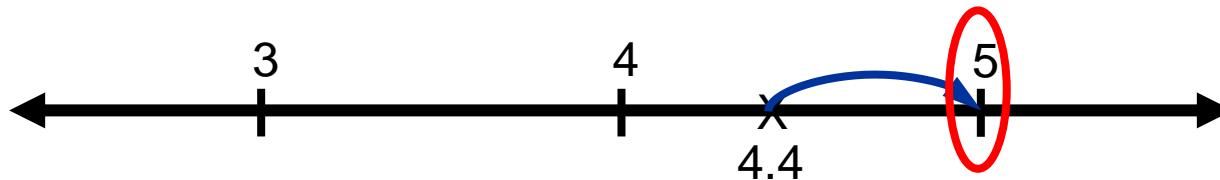
NewVar5	NewVar6
N 8	N 8
12.75	42.5

# The CEIL Function

The CEIL function returns the smallest integer greater than or equal to the argument.

General form of the CEIL function:

```
NewVar=CEIL(argument);
```



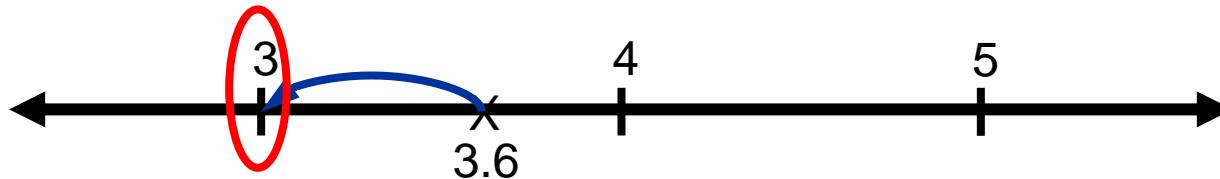
```
x=ceil(4.4);
```

# The FLOOR Function

The FLOOR function returns the greatest integer less than or equal to the argument.

General form of the FLOOR function:

```
NewVar=FLOOR(argument);
```



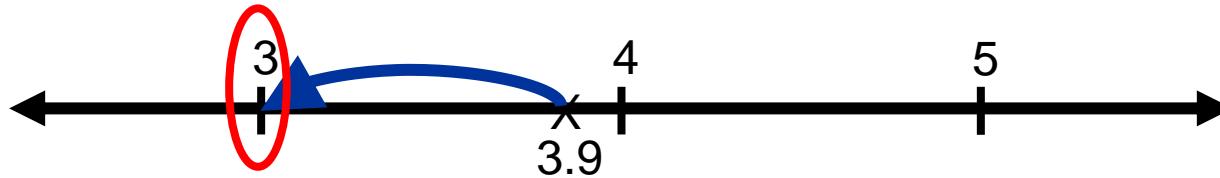
```
y=floor(3.6);
```

# The INT Function

The INT function returns the integer portion of the argument.

General form of the INT function:

```
NewVar=INT(argument);
```



```
z=int(3.9);
```

# Truncation Functions – Example

```
data truncate;  
  Var1=6.478;  
  CeilVar1=ceil(Var1);  
  FloorVar1=floor(Var1);  
  IntVar1=int(Var1);  
run;
```

PDV

Var1	CeilVar1	FloorVar1	IntVar1
6.478	7	6	6

# Setup for the Poll

In this program, the values returned from the FLOOR and INT functions are the same.

```
data truncate;
  Var1=6.478;
  CeilVar1=ceil(Var1);
  FloorVar1=floor(Var1);
  IntVar1=int(Var1);
run;
```

## PDV

Var1	CeilVar1	FloorVar1	IntVar1
6.478	7	6	6

# Poll

Given the same value as an argument, do the INT and the FLOOR functions always return the same result?

- Yes
- No

# Poll – Correct Answer

Given the same value as an argument, do the INT and the FLOOR functions always return the same result?

- Yes
- No



The INT and the FLOOR functions give different results if the argument value is negative.

# Truncation Functions

Compare the values from the CEIL, FLOOR, and INT functions with a negative argument.

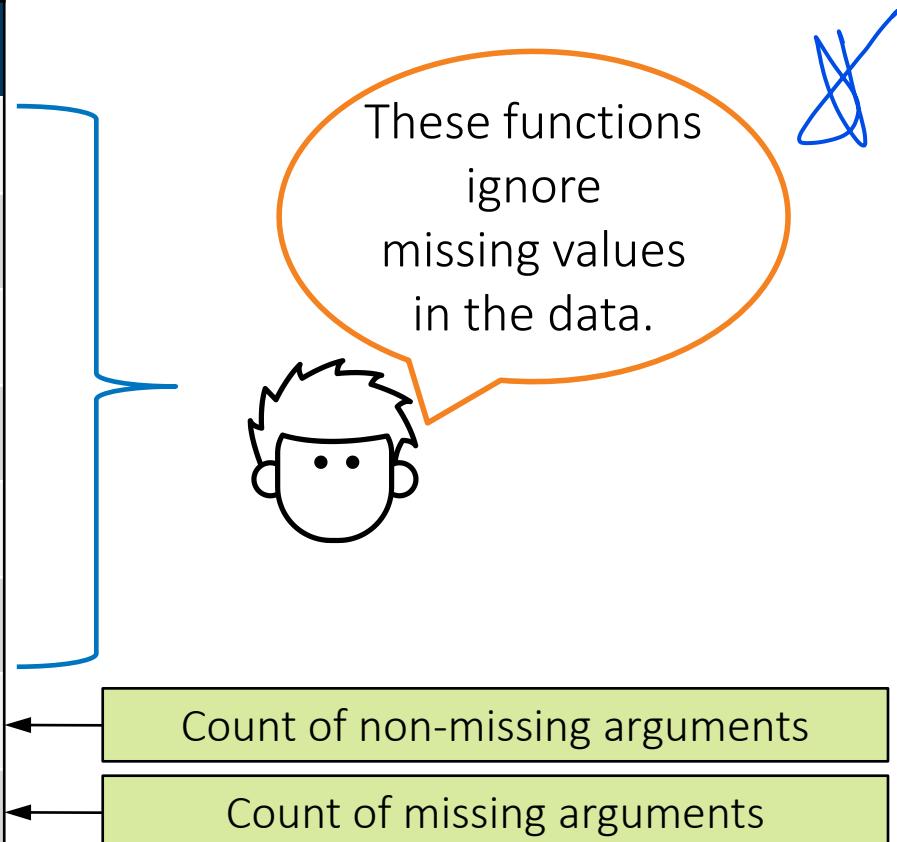
```
data truncate;  
  Var1=-6.478;  
  CeilVar1=ceil(Var1);  
  FloorVar1=floor(Var1);  
  IntVar1=int(Var1);  
run;
```

**PDV**

Var1	CeilVar1	FloorVar1	IntVar1
-6.478	-6	-7	-6

# Numeric Functions

Functions
SUM ( $num1, num2, \dots$ )
MEAN ( $num1, num2, \dots$ )
MEDIAN ( $num1, num2, \dots$ )
RANGE ( $num1, num2, \dots$ )
MIN ( $num1, num2, \dots$ )
MAX ( $num1, num2, \dots$ )
N ( $num1, num2, \dots$ )
NMISS   CMISS ( $num1, num2, \dots$ )



# Descriptive Statistics Functions

These functions all share the same general syntax:

*function-name(argument-1,argument-2,...,argument-n)*

- *argument-1* through *argument-n* are numeric. (except CMISS)
- An argument can be a variable list, which is preceded by OF.
- The non-counting functions ignore missing values in their arguments.

# Descriptive Statistics Functions

```
data descript;
  Var1=12;
  Var2=.;
  Var3=7;
  Var4=5;
  SumVars=sum(Var1,Var2,Var3,Var4);
  AvgVars=mean(of Var1-Var4);
  MissVars=cmiss(of Var1-Var4);
run;
```

## PDV

Var1	Var2	Var3	Var4
12	.	7	5

SumVars	AvgVars	MissVars
24	.	.

# Descriptive Statistics Functions

```
data descript;
  Var1=12;
  Var2=.;
  Var3=7;
  Var4=5;
  SumVars=sum(Var1,Var2,Var3,Var4);
  AvgVars=mean(of Var1-Var4);
  MissVars=cmiss(of Var1-Var4);
run;
```

## PDV

Var1	Var2	Var3	Var4
12	.	7	5

SumVars	AvgVars	MissVars
24	8	.

# Descriptive Statistics Functions

```
data descript;
  Var1=12;
  Var2=.;
  Var3=7;
  Var4=5;
  SumVars=sum(Var1,Var2,Var3,Var4);
  AvgVars=mean(of Var1-Var4);
  MissVars=cmiss(of Var1-Var4);
run;
```

## PDV

Var1	Var2	Var3	Var4
12	.	7	5

SumVars	AvgVars	MissVars
24	8	1

# Multiple Choice Poll

**Ord1**, **Ord2**, and **Ord3** are variables that contain the sale amounts of the last three orders from a customer. Which of the following expressions can calculate the total of the two largest orders?

- a. `sum(max(of Ord1-Ord3),max(of Ord1-Ord3))`
- b. `sum(of Ord1-Ord3)-min(of Ord1-Ord3)`
- c. `max(of Ord1-Ord3) + min(of Ord1-Ord3)`
- d. None of the above

# Multiple Choice Poll – Correct Answer

**Ord1**, **Ord2**, and **Ord3** are variables that contain the sale amounts of the last three orders from a customer. Which of the following expressions can calculate the total of the two largest orders?

- a. `sum(max(of Ord1-Ord3),max(of Ord1-Ord3))`
- b. `sum(of Ord1-Ord3)-min(of Ord1-Ord3)`
- c. `max(of Ord1-Ord3) + min(of Ord1-Ord3)`
- d. None of the above

**Adding the amount from all three orders and then subtracting the amount of the smallest order leaves the sum of the two largest orders.**