



RESOURCE GUIDE FOR THE COURSE ON



PROGRAMMING

COURSE CREATED AND INSTRUCTED BY:

ASLAM KHAN



[HTTPS://WWW.LINKEDIN.COM
/IN/ASLAMKHAN-PGMP-PMP/](https://www.linkedin.com/in/aslamkhan-pgmp-pmp/)



[@ASLAMKHAN54321](https://twitter.com/ASLAMKHAN54321)

COPYRIGHT MADE2STICK LEARNING

WWW.MADE2STICKLEARNING.COM



Legal Disclaimer:

The online course and this resource guide provided is not an official content from SAS Institute, nor is it affiliated with SAS Institute in any way. The course content is intended solely for educational purposes and is not to be reproduced or resold for commercial purposes.

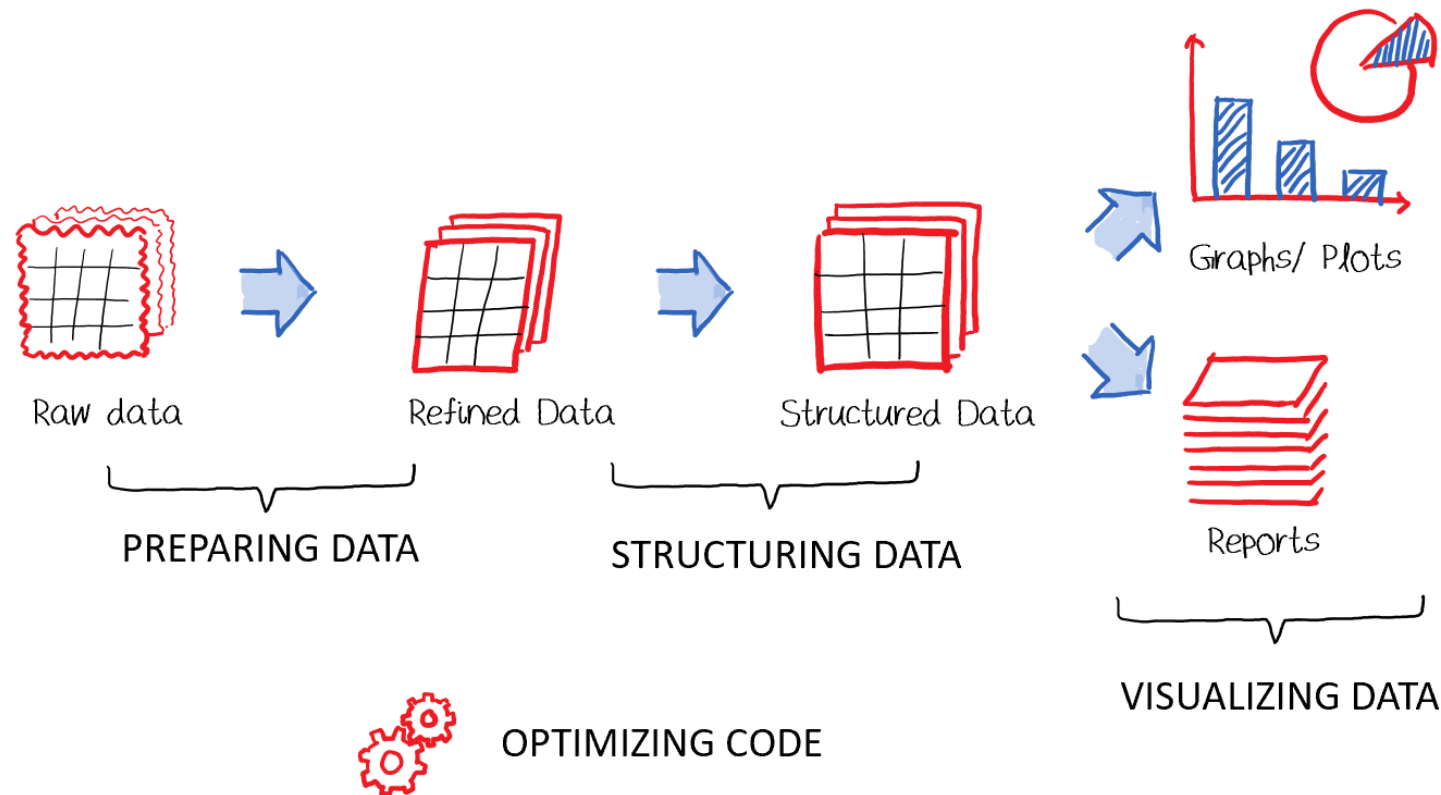
The information contained in the course is provided "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The course instructor shall not be liable for any direct, indirect, incidental, special, or consequential damages arising out of or relating to the use of or inability to use the course content or materials.

By accessing and using this resource guide, you acknowledge and agree to these terms and conditions.

Logo by SAS Institute - <http://www.sas.com>, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=5291445>

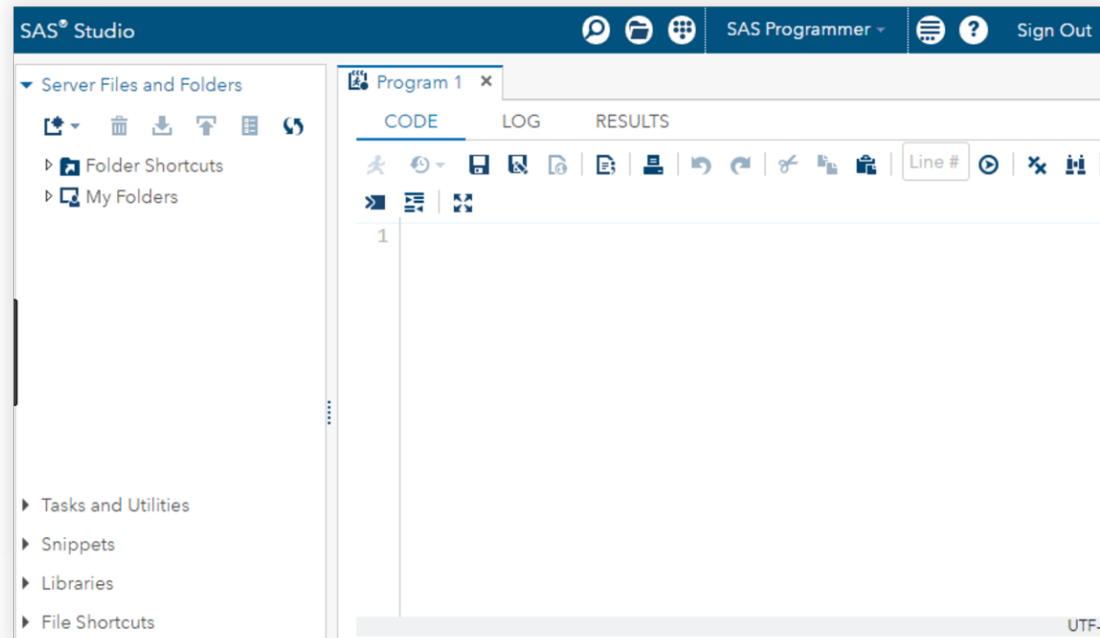
COURSE OVERVIEW



COURSE SECTIONS

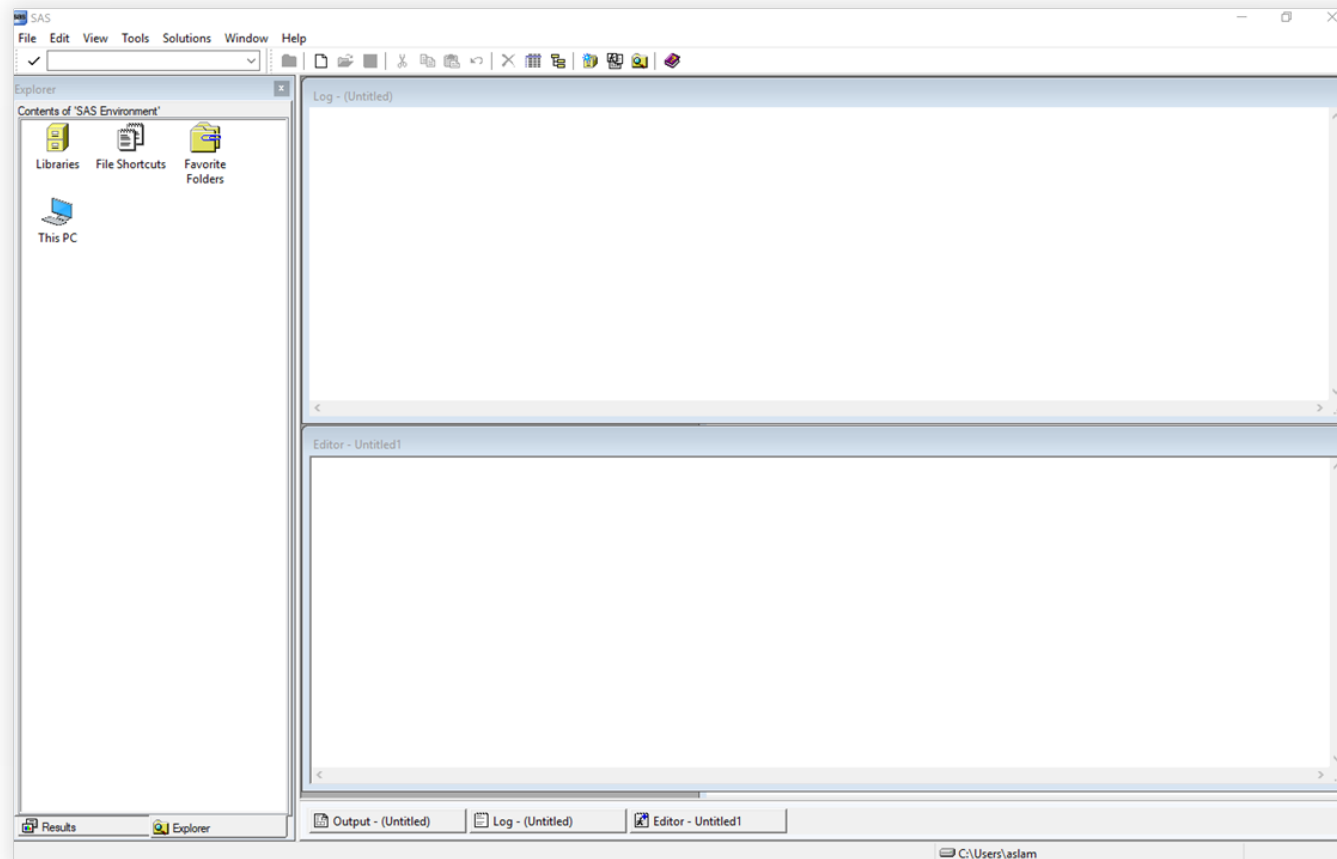
- 1.DATA PREPARATION** will teach you how to import data from multiple sources, create new variables, write SAS functions, and understand what goes on behind the scenes in SAS datasets
- 2.DATA STRUCTURING** will make you leap into transforming data to a new level by merging and joining multiple datasets together, or turning them upside-down (sorting), and side-ways (transposing)
- 3.DATA VISUALIZATION** will propel you further into the world of analytics and obtaining insightful inferences from what is inside your data
- 4.OPTIMIZING CODE** will take into the world of macro programming that teaches you how to write your code professionally and elegantly

SAS USER INTERFACE



-
- In this lecture, the various windows in SAS are introduced, including the Editor window where programming code is written, the Log window where SAS produces a log of code execution and error/warning/note messages, and the Explorer window for navigating data and libraries.
 - The Submit button, located in the Editor window, is used to submit code for execution. It is also mentioned that error messages in the Log window appear in red, warning messages appear in green, and notes appear in blue.
 - The Explorer window includes pre-installed libraries such as SASUSER and SASHELP, as well as the temporary Work library.
 - Data sets can be opened by double-clicking them in the Explorer window.
 - It is also mentioned that the Output window displays results from code execution.

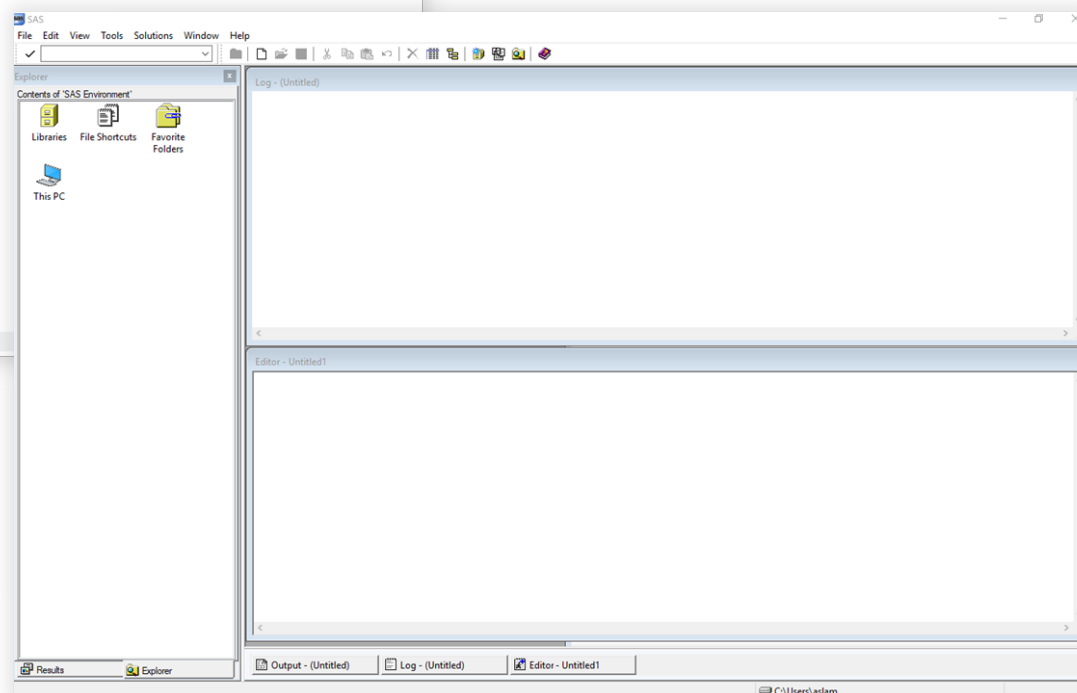
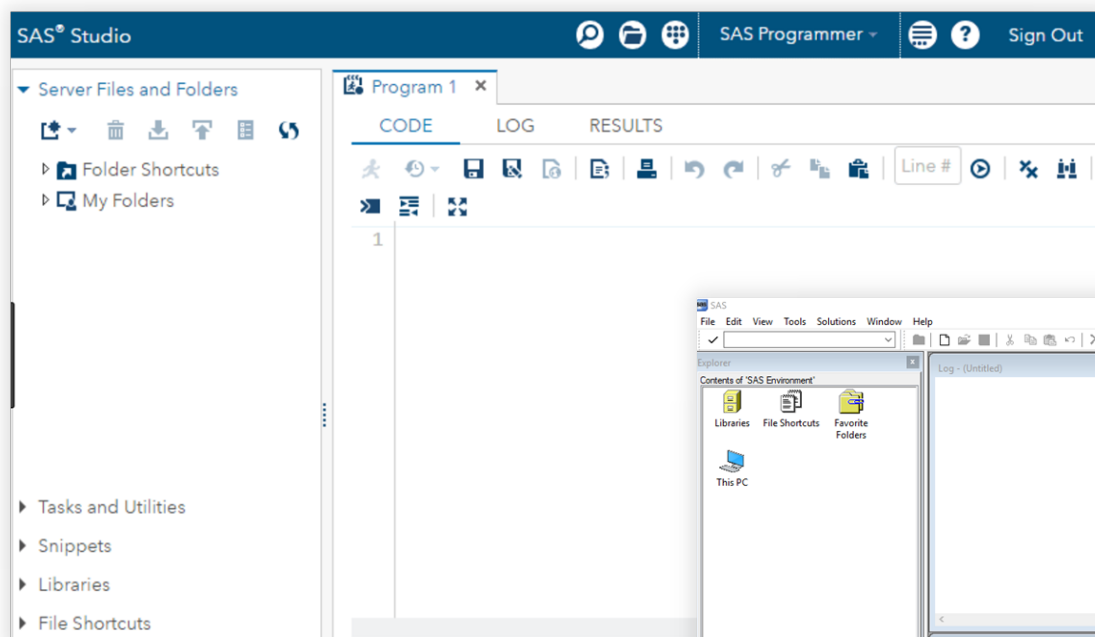
SAS USER INTERFACE – DESKTOP EDITION



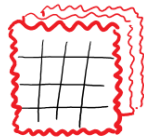
The 3 main windows are:

1. Explorer Window
2. Program Editor Window
3. Log Window

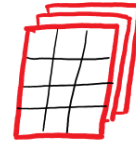
SAS USER INTERFACE COMPARISON



PREPARING DATA



Raw data



Refined Data

- ❑ SAS Dataset, Variables and observations
- ❑ Data Libraries
- ❑ Naming Conventions
- ❑ SAS Program syntax
- ❑ Data and Proc Steps
- ❑ Bringing data into SAS
- ❑ Program Data Vector (PDV)
- ❑ Creating new variables

- ❑ Keep, Drop and Rename variables
- ❑ If-else conditional statements
- ❑ Filtering data
- ❑ SAS Dates
- ❑ SAS Functions
- ❑ Format and informats
- ❑ Implicit and Explicit Output
- ❑ Do Loops

SAS DATASET

Variables

Observations

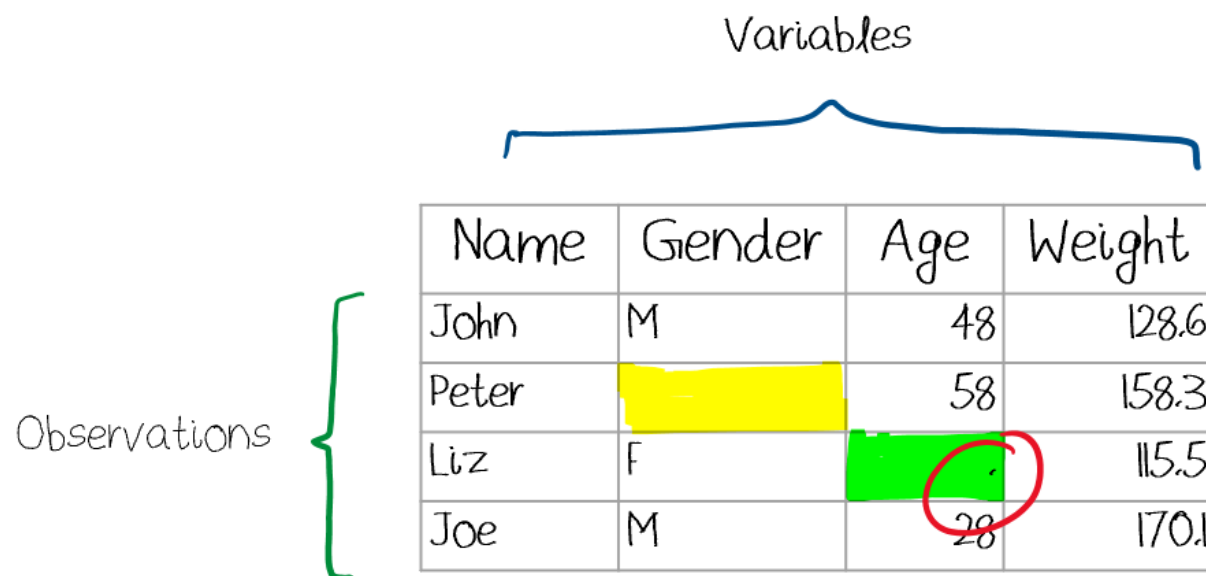
Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

- Rows and Columns
- Variables and observations
- Variable types – character and numeric

-
- In SAS programming, data is organized into tables called datasets that have rows and columns.
 - The columns are called variables and the rows are called observations.
 - There are two types of variables in SAS: character variables, which contain text-based data, and numeric variables, which contain numbers.
 - Character variables are left-aligned and numeric variables are right-aligned.

SAS DATASET

Variables



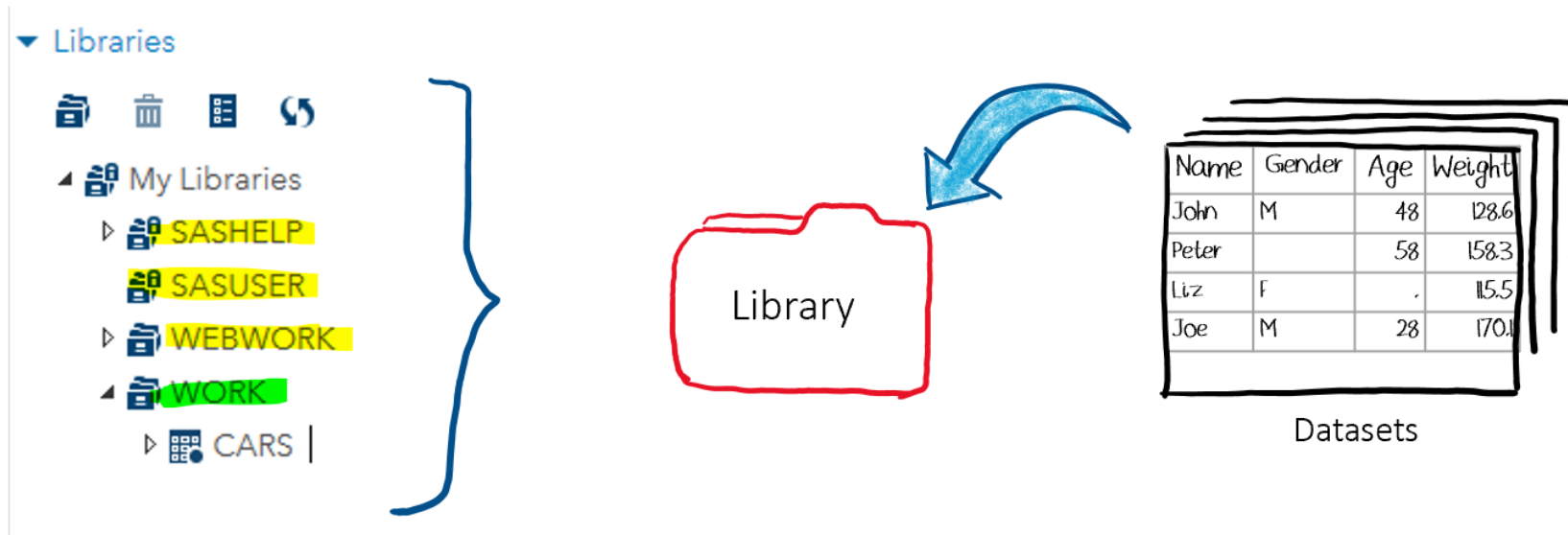
Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

Missing values

- Character variables are null
- Numeric variables are represented by a .

SAS handles missing values in character variables by representing them with a null value, while missing values in numeric variables are represented with a period or dot.

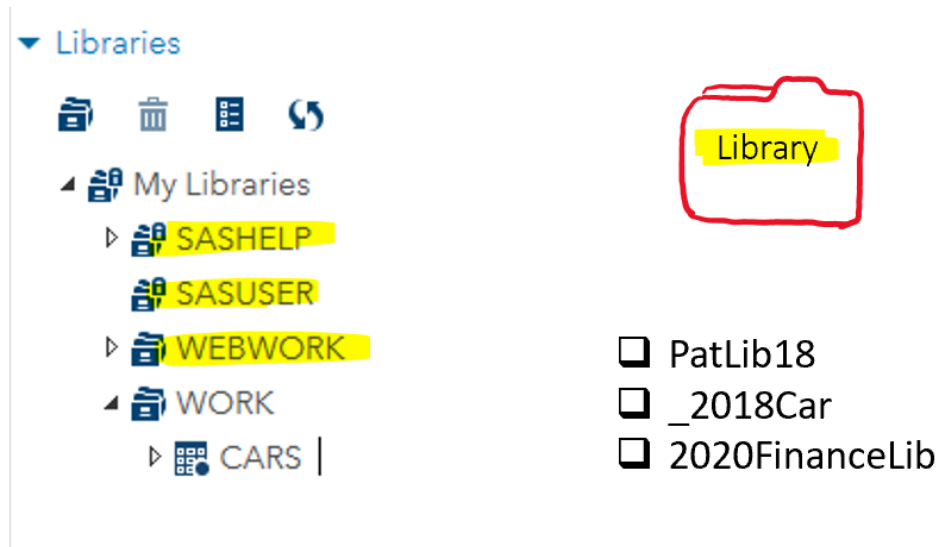
DATA LIBRARIES



Two types of libraries:

- Temporary
 - Permanent
-
- There are two main types of libraries: temporary and permanent.
 - Temporary libraries are located in a temporary folder in the computer's memory and the data stored there is lost once the session is closed.
 - Permanent libraries are stored on the computer or server and the reference to the data is retained even after the session is closed.
 - SAS provides sample permanent libraries, such as SAS Help and SAS User, which contain pre-made data sets that can be used for the coursework.
 - To view the contents of a data set in a permanent library, it can be opened by double clicking on it.
 - The data set's variables and observations will be displayed in a table, with character variables indicated by an icon and numeric variables indicated by a different icon.
 - If a data set spans multiple pages, it can be accessed by clicking on the arrows at the bottom of the table.

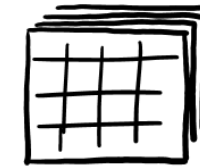
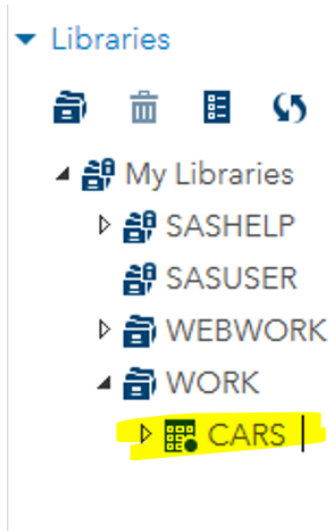
NAMING CONVENTIONS



- Can be 1 to 8 characters long
- Must begin with a letter (A-Z, either uppercase or lowercase) or an underscore (_)
- Can continue with any combination of numbers, letters, or underscores.

- The naming conventions for SAS libraries include having a name that is one to eight characters long, starting with a letter or underscore, and containing any combination of letters, numbers, or underscores after the first character.
- It is important to follow these conventions when creating your own SAS libraries.

NAMING CONVENTIONS



Datasets

- ☐ PatientData2018
- ☐ myBudgetThisYear
- ☐ _2018FinanceData
- ☐ 2018CarsSales

- ☐ Can be 1 to 32 characters long
- ☐ Must begin with a letter (A-Z, either uppercase or lowercase) or an underscore (_)
- ☐ Can continue with any combination of numbers, letters, or underscores.

There are three rules for naming conventions in SAS for both libraries and data sets.

- For libraries, the name must be between 1 and 8 characters in length, must begin with a letter or underscore, and can continue with any combination of numbers, letters, or underscores.
- For data sets, the name must be between 1 and 32 characters in length, must begin with a letter or underscore, and can continue with any combination of numbers, letters, or underscores.
- It is not allowed to use special characters like percentages, ampersands, plus signs, minus signs, stars, parentheses, dollars, or exclamation points in the names of libraries or data sets.
- If a name does not follow these rules, it will not be considered a valid SAS library or data set name.

SAS PROGRAM SYNTAX

```
data cars;  
    set sashelp.cars;  
run;
```

```
proc print data=cars;  
run;
```

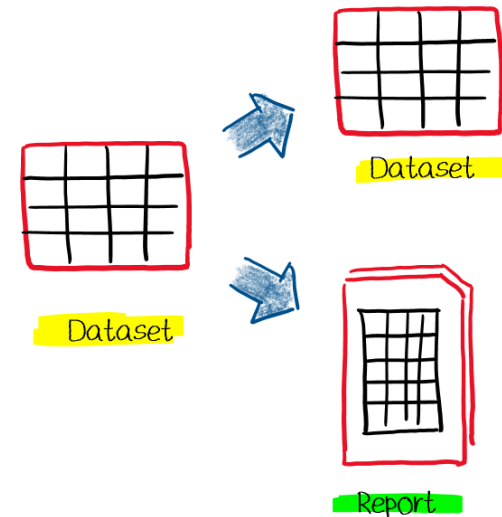
- It usually begins with a SAS keyword.
- It always ends with a semicolon ;
- SAS code is free text format, case insensitive, and can begin and end anywhere

-
- SAS is a programming language that uses simple English words to write code.
 - It has a specific syntax and structure, including using specific keywords and ending statements with semicolons.
 - SAS is case insensitive and can be written in multiple lines.
 - It also allows for spaces and line breaks in the code. Keywords are often colored in dark blue to make the code more readable.
 - SAS statements can begin and end anywhere in the programming editor.

DATA STEP AND PROC STEP

Data step { `data cars;`
 `set sashelp.cars;`
 `run;`

Proc step { `proc print data=cars;`
 `run;`

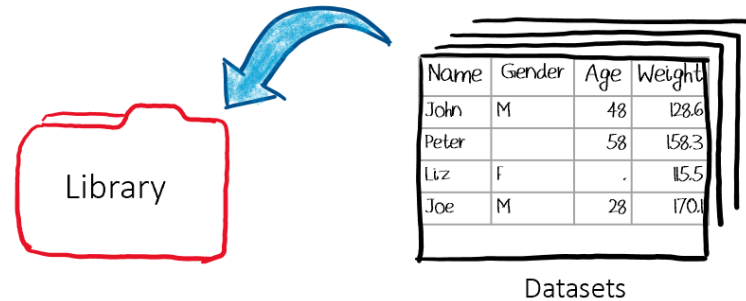


- In SAS programming, the source code is divided into two main modules: the data step and the proc step.
- The data step is a block of code that pertains to modifying a data set, such as its variables and observations, or transferring data from one place to another.
- The proc step is a block of code that performs an action with the data, such as printing, reporting, or sorting.
- A data step begins with the keyword "data" and ends with the keyword "run", while a proc step begins with the keyword "proc" and ends with the keyword "run".

REFERENCING DATA IN LIBRARIES

```
data cars;  
    set sashelp.cars;  
run;
```

```
proc print data=cars;  
run;
```



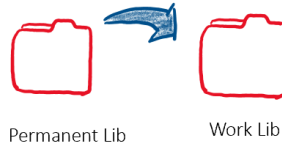
- Two level naming -> LibName.DatasetName
- Single level naming -> Library is **Work** by default if not explicitly mentioned

- Just like finding books in a library through a catalog, data can be called upon from somewhere inside a library in two ways.
- In SAS, there are two ways to reference data sets in libraries: two-level naming and single-level naming.
- In two-level naming, the data set name is preceded by the library name and a period, for example: "library.data set."
- Single-level naming does not include the library name, but it can only be used for data sets in the temporary library (called "work").
- To reference a data set in a permanent library, two-level naming must be used.

BRINGING DATA INTO SAS

From existing data within SAS

```
data cars;  
  set sashelp.cars;  
run;
```

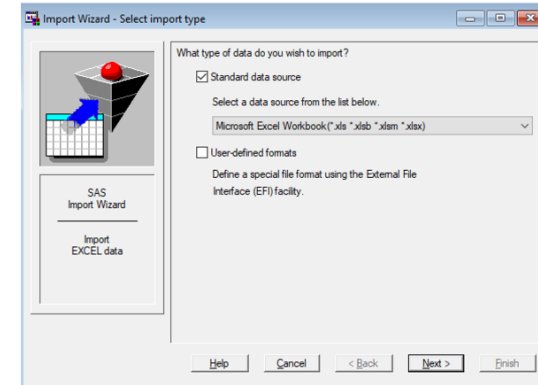


From creating your own data within SAS

```
data patients;  
  input name $ gender $ age weight;  
  cards;  
  John M 48 128.6  
  Peter M 58 158.3  
  Liz F 45 115.5  
  Joe M 28 170.1  
  ;  
run;
```



By importing data from Excel spreadsheets outside SAS



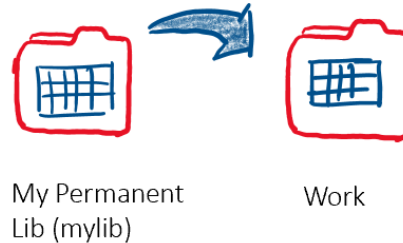
In summary, there are three methods of bringing data into SAS. The first method is using existing data within SAS, such as data in the permanent libraries. The second method is creating data within the SAS programming window. The third method is importing data from external sources, such as Excel spreadsheets or other databases.

The different methods for bringing data into SAS are discussed here.

- The first method is using existing data in SAS, which is stored in permanent libraries. This can be done through a data step in SAS, which involves specifying the data set name and the library it is coming from.
- The second method is creating data within the SAS programming window and storing it in either permanent or temporary libraries. This is also done through a data step, using the input statement to specify the variable names and using the datalines statement to input the data.
- The third method is importing data from external sources, such as Excel sheets or databases. This can be done using the proc import statement, which allows you to specify the file path and file type of the data. There are other methods to import data which can be done using the infile and input statements.

BRINGING DATA INTO SAS

From existing data outside of SAS
Using **LIBNAME** statement

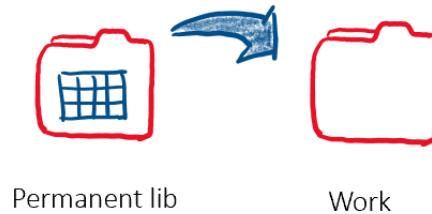


```
libname mylib '/folders/myshortcuts/myFolders';  
data class;  
set mylib.class;  
run;
```

- In SAS, a libname is a reference to a SAS library, which is a collection of SAS data sets and/or other files.
- The libname statement assigns a SAS library reference to a physical location on a storage device, such as a disk drive or a directory on a server.
- Once you have defined a libname for a SAS library, you can use the libname as a prefix to refer to the data sets and files stored in that library.
- In this example, the libname statement assigns the SAS library reference mylib to the directory in between the quotes.
- The data statement then creates a new SAS data set called class in the work library, and the set statement reads in data from the class data set in the mylib library.

BRINGING DATA INTO SAS

From delimited data outside SAS
Using **FILENAME** statement

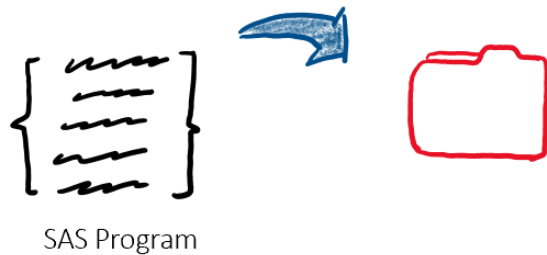


```
filename myclass '/folders/myshortcuts/myFolders/class.dat';  
data class;  
infile myclass;  
input name $ gender $ age weight;  
run;
```

-
- The filename statement is used to assign a fileref (short for file reference) to a physical file on a storage device.
 - A fileref is a symbolic name that represents the location of a file, and it can be used as a shorthand way of referring to the file in SAS programming statements.
 - In this example, the filename statement assigns the fileref myclass to the file '.../class.dat'.
 - The infile statement in the data step then reads data from the file using the myclass fileref.

BRINGING DATA INTO SAS

From Delimited data with SAS



```
data class;  
infile cards delimiter =',';  
input name $ gender $ age weight;  
cards;  
John,M,48,128.6  
Peter,,58,158.3  
Liz,F,,115.5  
Joe,M,28,170.1  
;  
run;
```

- To bring data into SAS using the delimiter option, you will need to use the infile statement in a data step to read the data.
- The infile statement has a delimiter option that allows you to specify a character or symbol that is used to separate columns in the data.
- In this example, the infile statement in the data step reads data from the cards statement using the delimiter option to specify that the data is separated by commas.
- The input statement reads three variables, name gender age and weight, from the data, and the cards statement provides the data values.

BRINGING DATA INTO SAS

From Delimited data with SAS



```
data class;  
infile cards delimiter =',' dsd;  
input name $ gender $ age weight;  
cards;  
"John,K",M,48,128.6  
"Peter,P",.,58,158.3  
"Liz,W",F,.,115.5  
"Joe,R",M,28,170.1  
;  
run;
```

- The infile statement in the data step reads data from the cards statement using the delimiter option to specify that the data is separated by commas and the dsd option to tell SAS to treat consecutive delimiters as a single delimiter.
- The input statement reads 4 variables, from the data, and the cards statement provides the data values.

BRINGING DATA INTO SAS

Column Input



SAS Program



```
data class;  
input name $ 1-5 age 6-7 gender $ 8 weight 9-15;  
cards;  
John 48M128.6  
Peter58.158.3  
Liz . F115.5  
Joe 28M170.1  
;  
run;
```

SAS provides several methods for importing data, including column input, list input, and pointer input.

- To bring data into SAS using column input with the cards statement, you will need to use the infile statement in a data step to read the data. The infile statement has a column option that allows you to specify the starting and ending positions of each column in the data. You can use the input statement to read the data from the specified columns into SAS variables. In this example, the infile statement in the data step reads data from the cards statement. The input statement specifies the starting and ending positions of each column in the data using the column option. The variables name, age, gender, and weight are read from the specified columns in the data, and the cards statement provides the data values.
- The @ symbol can be used as a pointer to specify the starting position of a column of data. The @ symbol is used to specify that the data in the last column starts at the 17th position. The dollar sign after the column gender specification indicates that the column contains character data. The input statement reads in the data from the "cards" statement, which specifies the actual data being imported.
- you can use line input to bring in data by specifying the starting and ending positions of the data in each line. The # symbol can be used as a pointer to specify the starting position of a line of data. In our example shown, the code brings in data using line input and the # symbol to specify the starting position of each line.
- The trailing @ symbol in SAS is used in the input statement to indicate that the rest of the line should be treated as a single variable. This is often used when the data being imported contains multiple lines for each record.
- The double trailing @@ symbol in SAS is used in the input statement to indicate that the rest of the input lines should be treated as a single variable. This is often used when the data being imported contains multiple lines for each record

BRINGING DATA INTO SAS

Proc Import from Excel sheets



To bring data into SAS from an Excel file using the proc import procedure, you can use the following syntax:

```
proc import datafile="...\class.xlsx" out=class dbms=excel getnames=yes; run;
```

- In this example, the "datafile" option is used to specify the path to the Excel file that is being imported.
- The "out" option is used to specify the name of the SAS dataset where the imported data will be stored.
- The "dbms" option is used to specify the type of data being imported, in this case Excel data. The "getnames" option is used to specify that the first row of the Excel file should be treated as variable names.

After running this code, a SAS dataset called "class" will be created, containing the data from the specified Excel file. The variable names for the dataset will be taken from the first row of the Excel file.

BRINGING DATA INTO SAS

Proc Import from Txt files



The proc import procedure in SAS can be used to import data from text files that are formatted in different ways. For example, data can be separated by commas or by spaces. The proc import syntax remains the same, but the "delimiter" option can be used to specify the character that separates the data points in the text file.

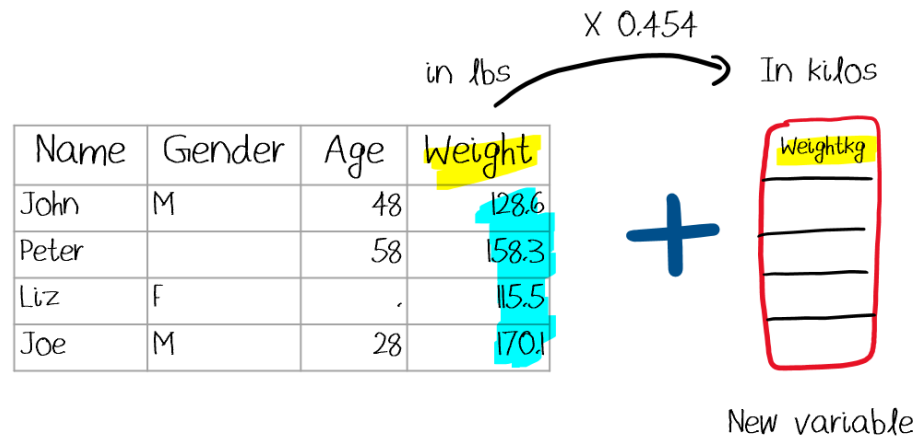
To bring data into SAS from an Excel file using the proc import procedure, you can use the following syntax:

```
proc import datafile="...\class.txt" out=class dbms=excel getnames=yes; run;
```

In this example, the "datafile" option is used to specify the path to the TXT file that is being imported. The "out" option is used to specify the name of the SAS dataset where the imported data will be stored. The "dbms" option is used to specify the type of data being imported, in this case Excel data. The "getnames" option is used to specify that the first row of the Excel file should be treated as variable names.

After running this code, a SAS dataset called "class" will be created, containing the data from the specified Excel file. The variable names for the dataset will be taken from the first row of the TXT file.

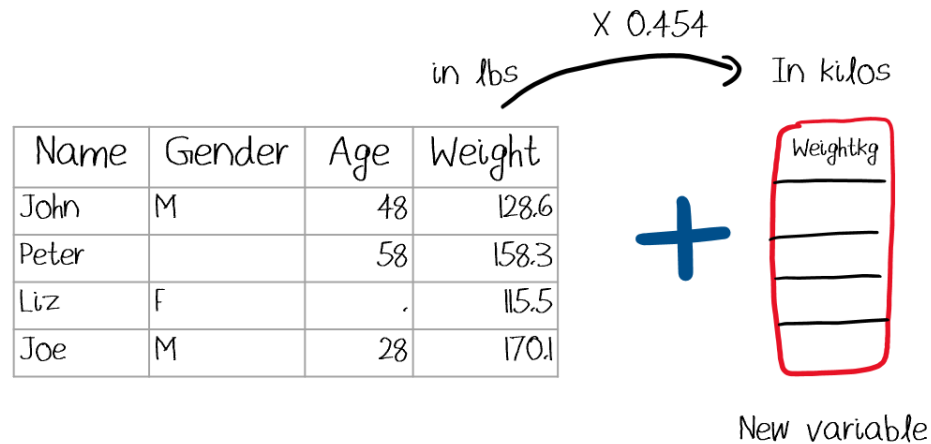
CREATING NEW VARIABLES



```
data class;  
  set sashelp.class;  
  
  weightkg = weight * 0.454;  
  
run;
```

- To create a new variable in SAS, you can use the DATA step.
- First, you will need to define the dataset that you want to create the new variable in.
- Then, you can create the new variable by specifying its name followed by an equal sign and the definition for the new variable.
- The definition can be based on existing variables in the dataset or can be a completely new and independent value.
- You can also use various functions and operators to modify the values of the new variable.
- Finally, you can use the RUN statement to execute the DATA step and create the new variable in the dataset.

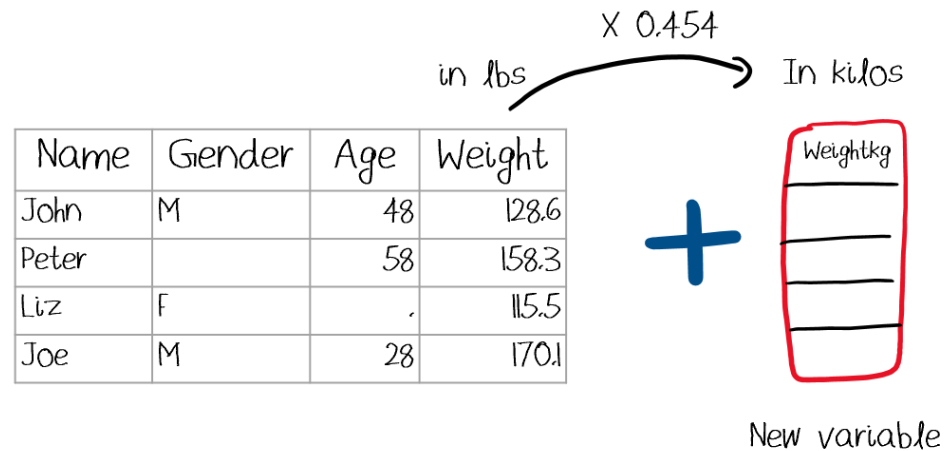
KEEP STATEMENT



```
data class;  
    set sashelp.class;  
  
    weightkg = weight * 0.454;  
    Keep name gender age weightkg;  
  
run;
```

- The KEEP statement in SAS is used to specify the variables that you want to include in your final dataset.
- It is used to exclude variables that are not needed in the final dataset.
- The KEEP statement is written before the RUN statement and is followed by a list of variables that you want to include in the final dataset.
- For example, if you have a dataset called class that contains variables name, sex, age, weight and weight kg, and you only want to keep variables name, sex, age, weightkg in your final dataset, you would write the following code: KEEP name sex age weightkg ;
- This will create a new dataset with only variables name sex age weightkg , and will exclude variable weight.

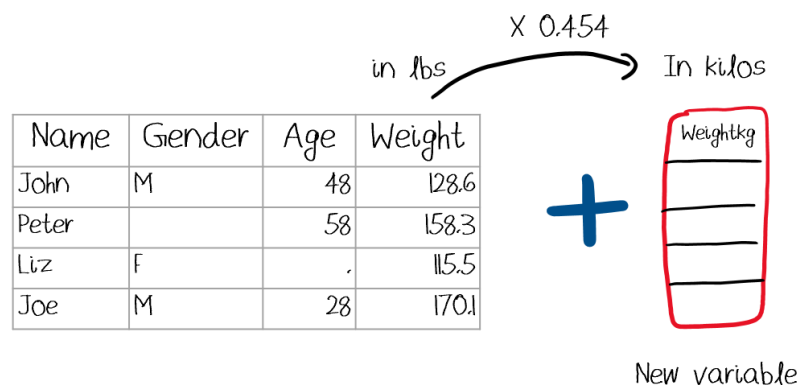
DROP STATEMENT



```
data class;  
    set sashelp.class;  
  
    weightkg = weight * 0.454;  
    Drop weight;  
  
run;
```

- The drop statement in SAS is used to remove variables from a dataset.
- It is typically used in a data step to specify which variables should be excluded from the output dataset.
- The drop statement is used by specifying the name of the variable(s) to be removed, separated by a space.
- In this example, the drop statement is used to remove the variable weight from the dataset sashelp.class before creating the new dataset class.
- After the drop statement is executed, the final dataset will not contain the variables that were specified in the drop statement.

RENAME STATEMENT



```
data class;  
    set sashelp.class;  
  
    weightkg = weight * 0.454;  
    Rename sex= gender;  
  
run;
```

- The rename statement in SAS is used to change the name of a variable in a dataset.
- In this case, the rename statement is being used to change the name of the variable "sex" to "gender".
- The syntax for using the rename statement is as follows:

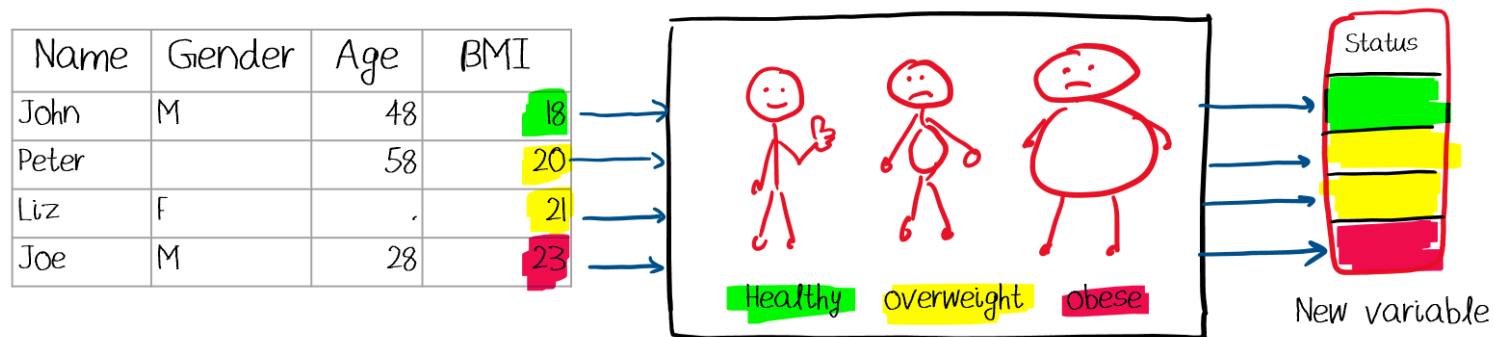
rename variable-name = new-name;

For example, to rename the variable "sex" to "gender", the following statement would be used:

rename sex = gender;

The rename statement can be used in combination with other statements, such as the keep and drop statements, to further modify a dataset.

CONDITIONAL STATEMENTS



If-else conditional statements

```
data class;  
    set sashelp.class;  
...  
if bmi<=18 then status='Healthy weight';  
else if 18<bmi<=21 then status='Overweight';  
else if bmi> 21 then status='Obese';  
  
run;
```



In SAS, the if statement is used to execute code conditionally based on a specified criteria. The condition is a logical expression that evaluates to either true or false. If the condition is true, the code in the then block will be executed. If the condition is false, the code in the else block will be executed.

To use if-else conditional logic in SAS, you can use the following syntax:

```
if (condition) then action1; else if (condition) then action2; else action3;
```

For example, to calculate a status variable based on BMI values where the status results in "Healthy weight" if the BMI is less than 18, "Overweight" when the BMI is between 18 and 21, and "Obese" when the BMI is greater than 21, you can use the following code:

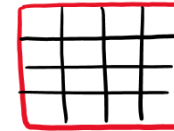
```
if (bmi < 18) then status = "Healthy weight"; else if (bmi >= 18 and bmi <= 21) then status = "Overweight"; else status = "Obese";
```

This code will evaluate the condition in the "if" statement first. If the condition is true, the action following the "then" statement will be executed (in this case, setting the value of the "status" variable to "Healthy weight"). If the condition in the "if" statement is false, the code will move on to the "else if" statement and evaluate the second condition. If the second condition is true, the action following the "then" statement will be executed (in this case, setting the value of the "status" variable to "Overweight"). If both the "if" and "else if" conditions are false, the code will execute the action following the "else" statement (in this case, setting the value of the "status" variable to "Obese").

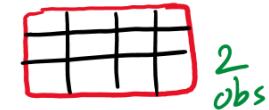
FILTERING DATA

Data
step

```
data class;  
set sashelp.class;  
where sex='F';  
run;
```

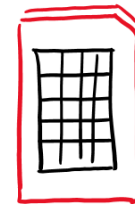


→
Data
step

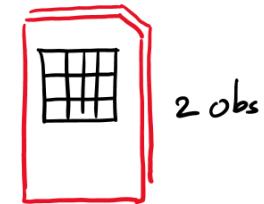


Proc
step

```
proc print data=sashelp.class;  
where sex='F';  
run;
```



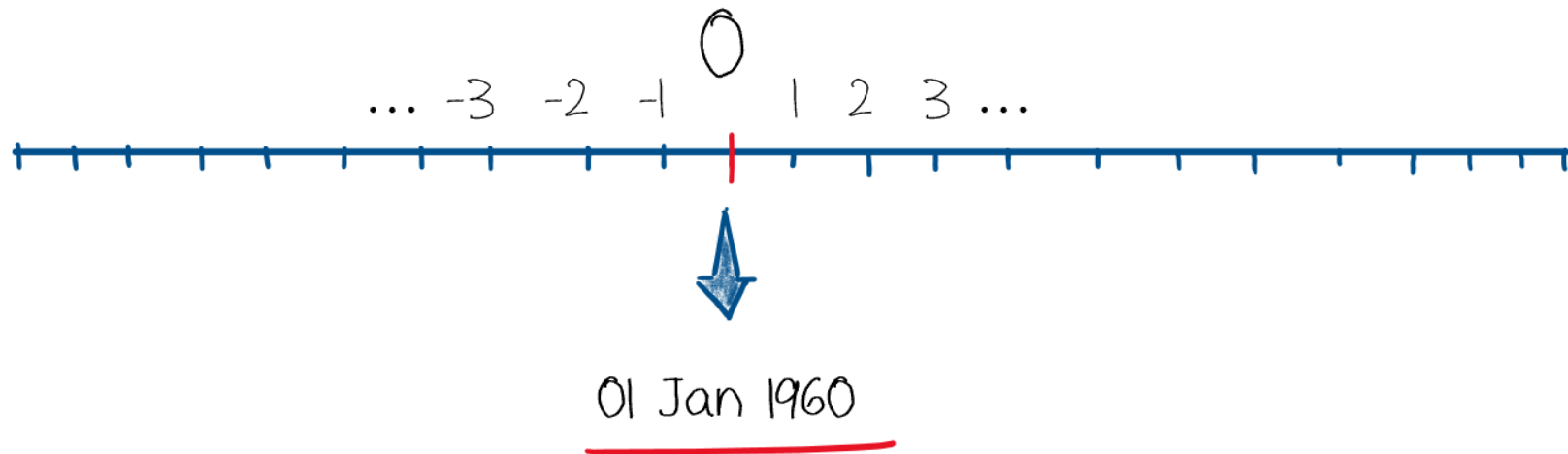
→
proc
step



With filter

- Filtering in SAS refers to the process of selecting a subset of observations from a dataset based on a specified criteria.
- It can be applied to both a data step and a proc step, but the way it works is different in each case. In a data step, filtering reduces the number of observations in the resulting dataset. In a proc step, it reduces the number of observations in the report output but does not affect the underlying dataset.
- Filtering is done using the WHERE statement, which specifies the criteria for selecting the observations.
- The syntax for filtering in a data step or proc step is the same and involves using the WHERE statement followed by the criteria for selection.
- For example, the WHERE statement "WHERE sex='F'" would select only observations with a value of 'F' in the sex variable.

SAS DATES



E.g. 31 Dec 2018 = 21549

SAS DATES

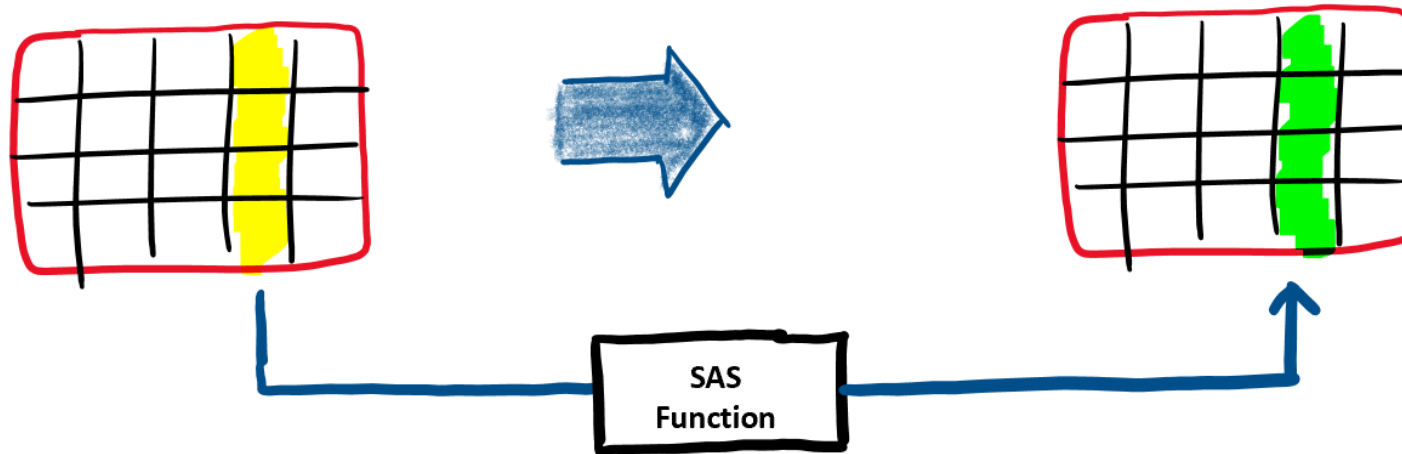
In SAS, dates are values that represent a specific point in time. They are stored in a numerical format, with 1st January 1960 being given a value of zero and each subsequent day having a value one higher than the previous day. This numerical representation allows for arithmetic to be performed on dates, such as finding the difference between two dates. Dates can be written in different formats in SAS, such as "31 December 18" or "20-18-12-31" (YYYY-MM-DD). In SAS, the suffix "D" is used to indicate that a value is a date in numerical format. It is important to note that SAS handles dates differently than other programming languages.

A date format is a way to specify how a date value should be displayed or written as a character string. SAS supports a wide range of date formats, which can be used to display dates in different ways, such as with the month written as a name or as a number, or with the year written with four digits or two digits.

Here are some examples of SAS date formats:

- ddmmyy - displays the date as a three-letter abbreviation of the month name followed by the day and year (e.g. "14JAN97")
- ddmmy - displays the date as the day, month, and year with two digits each (e.g. "140197")
- ddmonyy - displays the date as the day, first three letters of the month name, and year (e.g. "14JAN1997")
- ddmonyy - displays the date as the day, first three letters of the month name, and last two digits of the year (e.g. "14JAN97")
- ddmmyyyy - displays the date as the day, month, and year with two digits for the month and four digits for the year (e.g. "14011997")
- mmddyy - displays the date as the month, day, and year with two digits each (e.g. "011497")
- mmddy - displays the date as the month, day, and year with two digits for the month and four digits for the year (e.g. "01141997")
- These are just a few examples of the many date formats available in SAS. You can use the FORMAT statement to specify a date format for a SAS variable.

FUNCTIONS



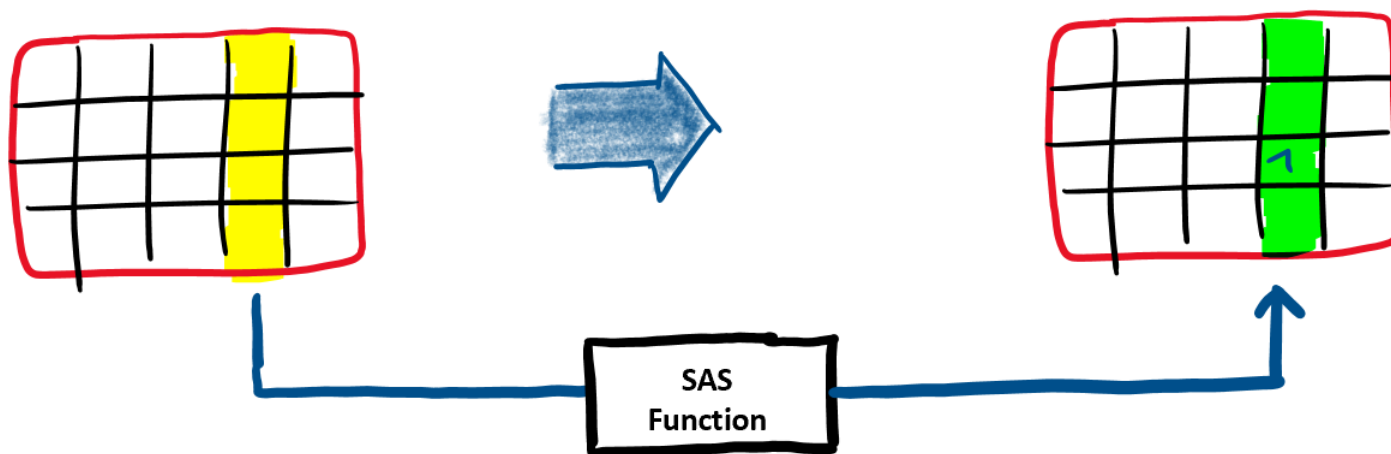
■ Functions alter a particular variable to new values

- Character functions
- Numeric functions

- SAS functions are used to modify variables to take on new values.
- They can be applied to one variable or a group of variables and can be used to modify character or numeric variables.
- There are two types of functions in SAS: character functions and numeric functions.
- Character functions perform modifications on character variables and usually result in a new character variable.
- Numeric functions perform modifications on numeric variables and usually result in a new numeric variable.
- SAS functions have a syntax similar to creating a new variable, with the name of the function followed by parentheses enclosing the variables to be modified.

CHARACTER FUNCTIONS

Case change



Character Functions

- Uppcase
- Lowcase
- Propcase

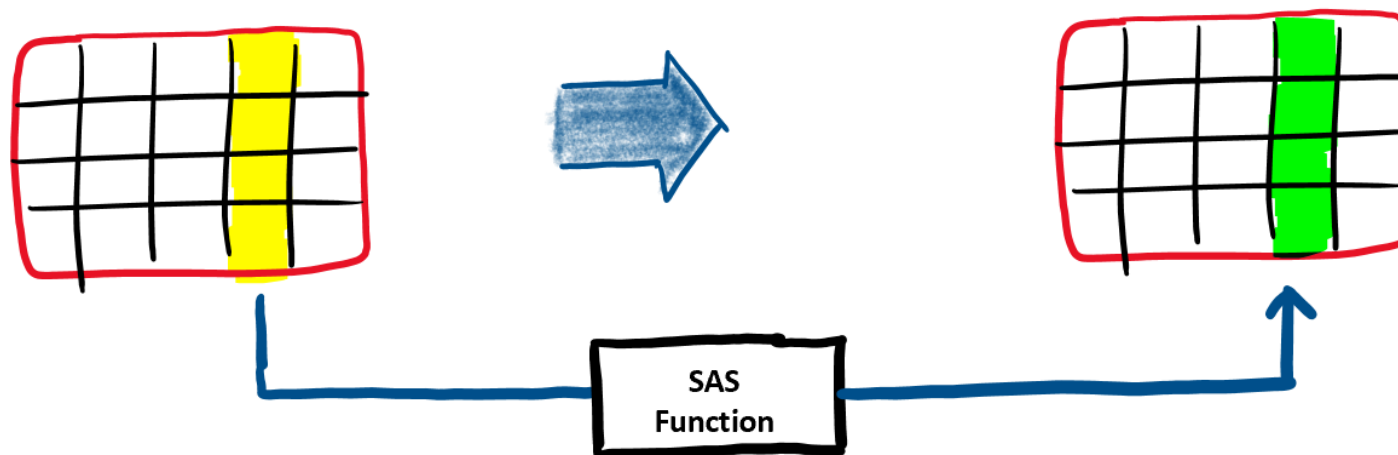
Character functions are functions that perform some kind of modification on a character variable and usually result in a new character variable with modified values. There are various character functions available in SAS, including functions for manipulating strings, extracting substrings, and converting between character and numeric data types.

Here are some examples of character functions in SAS:

- UPCASE - converts a character variable to uppercase
- LOWCASE - converts a character variable to lowercase
- PROPCASE - converts a character variable to propercase

CHARACTER FUNCTIONS

Length



Character Functions

- Length

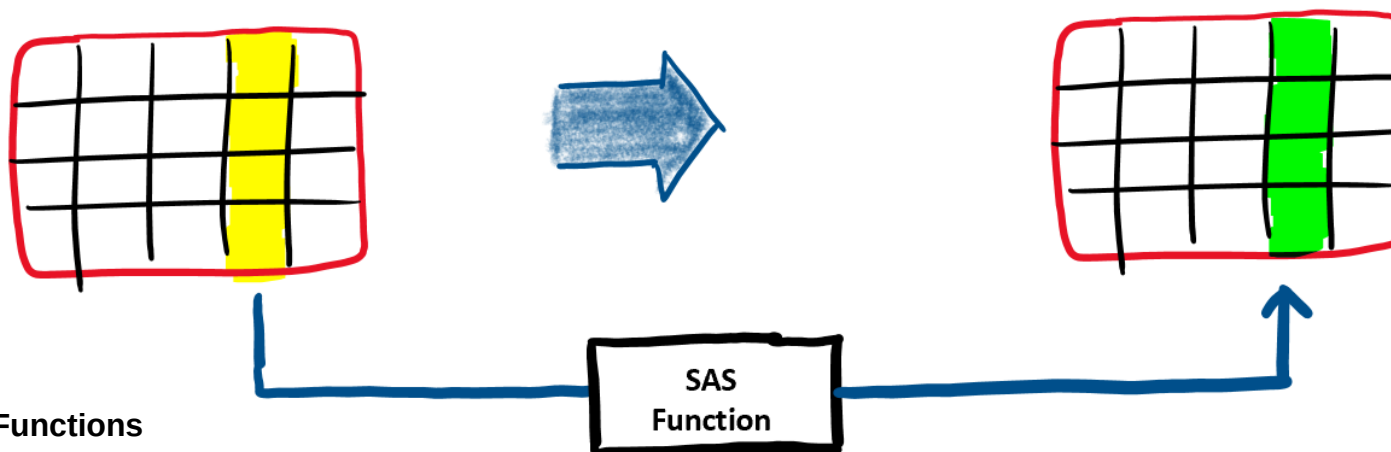
LENGTH(string)

Where "string" is the character value for which you want to find the length.

- The length function in SAS is used to find the number of characters in a particular character variable.
- It takes a character variable as an input and returns an integer value representing the length of that variable.
- For example, if a character variable called "name" has the value "John", the length function applied to "name" would return 4.
- If the character variable has spaces or other special characters, they are also included in the length calculation.
- The length function can be useful in situations where you need to know the size or length of a character variable for comparison or other purposes.

CHARACTER FUNCTIONS

Combining strings



Character Functions

- Cat

```
new_variable = CAT(char1, char2, ...);
```

where "new_variable" is the name of the new character variable that will be created, and "char1", "char2", etc. are the names of the character variables that you want to combine

The CAT function in SAS is a character function that concatenates or combines the values of two or more character variables and returns a new character value. The syntax for the CAT function is:

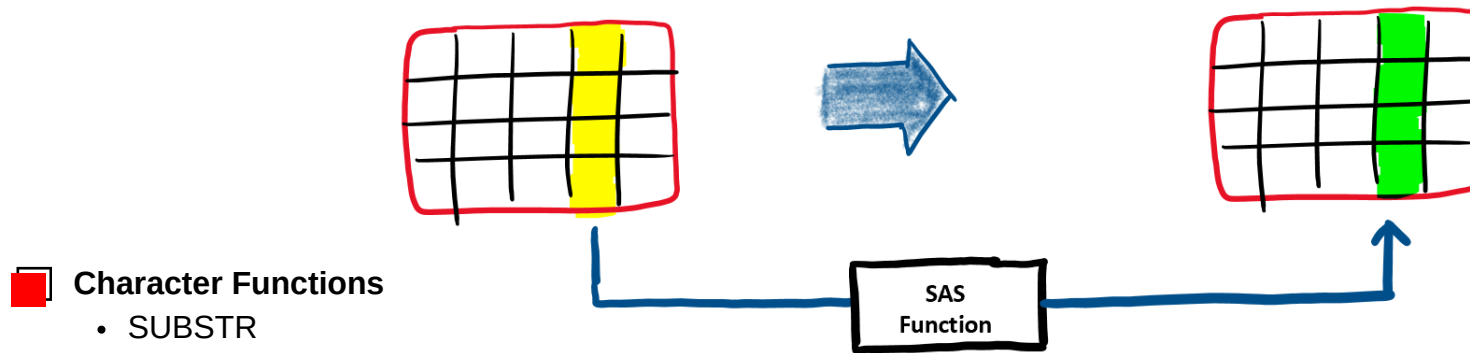
```
new_variable = CAT(char1, char2, ...);
```

where "new_variable" is the name of the new character variable that will be created, and "char1", "char2", etc. are the names of the character variables that you want to combine.

The values of these character variables will be combined in the order that they are listed, with no spaces or separators between them. For example, if char1 has a value of "John" and char2 has a value of "Doe", the new variable created by the CAT function will have a value of "JohnDoe".

CHARACTER FUNCTIONS

Substring



Character Functions

- SUBSTR

SUBSTR(string, start, length)

- "string" is the character string from which the substring will be extracted.
- "start" is the position of the first character in the substring. The position is specified as an integer and starts at 1 for the first character in the string.
- "length" is the number of characters to include in the substring. If "length" is not specified, the function extracts all characters from the start position to the end of the string.

The **SUBSTR** function in SAS extracts a substring from a character string. The function has the following syntax:

SUBSTR(string, start, length)

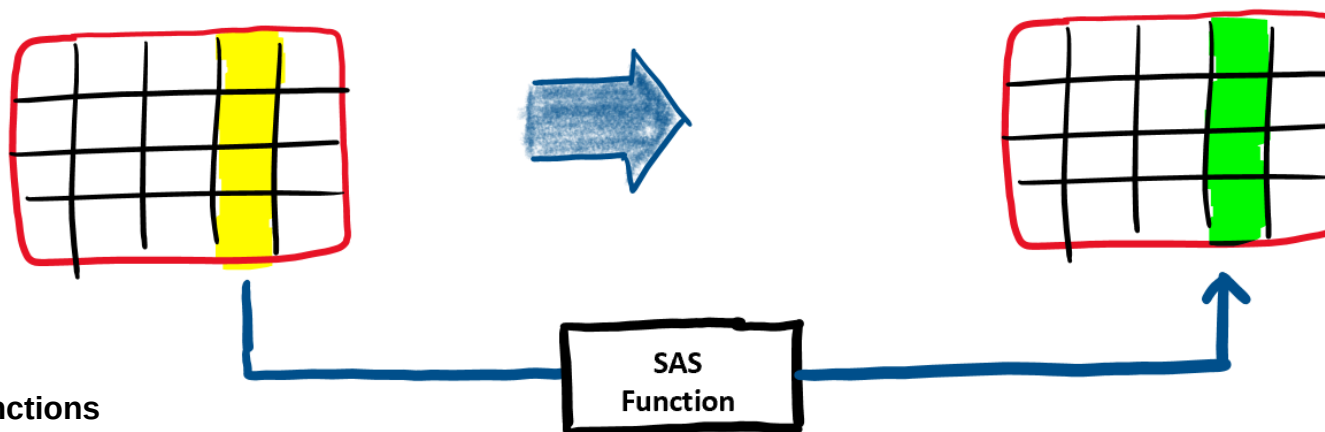
- "string" is the character string from which the substring will be extracted.
- "start" is the position of the first character in the substring. The position is specified as an integer and starts at 1 for the first character in the string.
- "length" is the number of characters to include in the substring. If "length" is not specified, the function extracts all characters from the start position to the end of the string.

For example, if "string" is "abcdef", the following calls to the SUBSTR function would produce the following results:

- SUBSTR("abcdef", 1, 2) returns "ab"
- SUBSTR("abcdef", 3, 3) returns "cde"
- SUBSTR("abcdef", 4) returns "def"

CHARACTER FUNCTIONS

TRIM



Character Functions

- TRIM

TRIM(string)

where "string" is the character value or variable that you want to trim.

The TRIM function in SAS is a character function that is used to remove leading or trailing spaces from a character value. It takes a character value as an input and returns a modified character value with the leading or trailing spaces removed. The function has the following syntax:

TRIM(character-value)

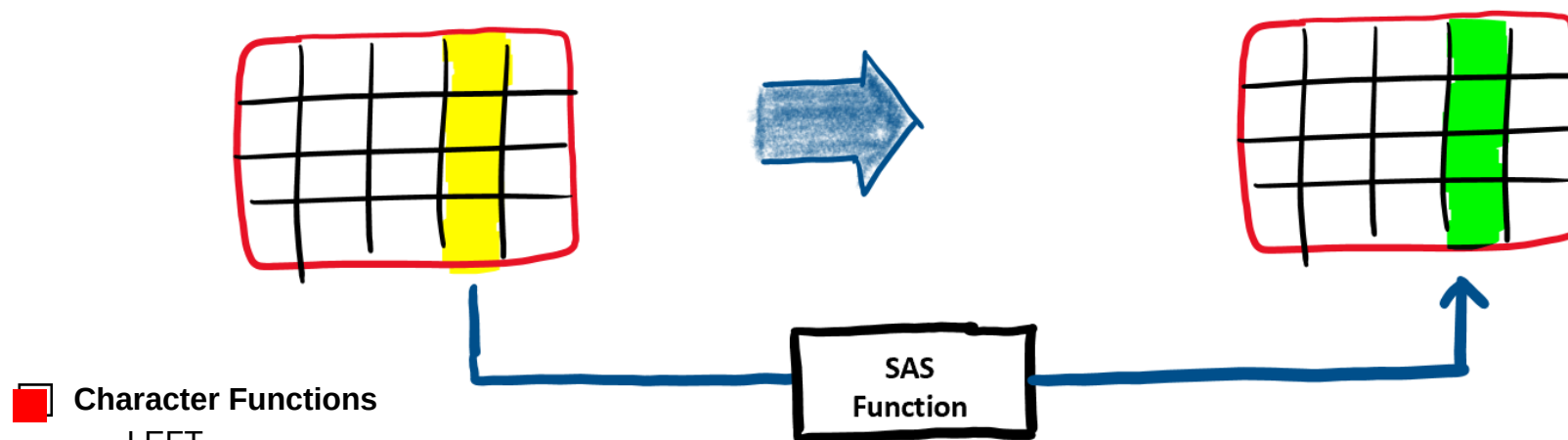
For example, if you have a character value ' Hello ' with leading and trailing spaces, you can use the TRIM function to remove the spaces like this:

```
result = TRIM(' Hello ');
```

The resulting value of 'result' would be 'Hello', with all leading and trailing spaces removed. The TRIM function is often used to clean up data that has been imported from external sources, where values may have extra spaces due to formatting or other issues.

CHARACTER FUNCTIONS

LEFT



LEFT(string, n)

where:

- 'string' is the character string that you want to extract the leftmost characters from.
- 'n' is the number of characters to extract from the left side of the string.

The LEFT function in SAS is a character function that extracts a specified number of characters from the left side of a character string. For example, if you have a character string 'Hello World' and you apply the LEFT function to extract the first 3 characters, the resulting value would be 'Hel'. The syntax for the LEFT function is as follows:

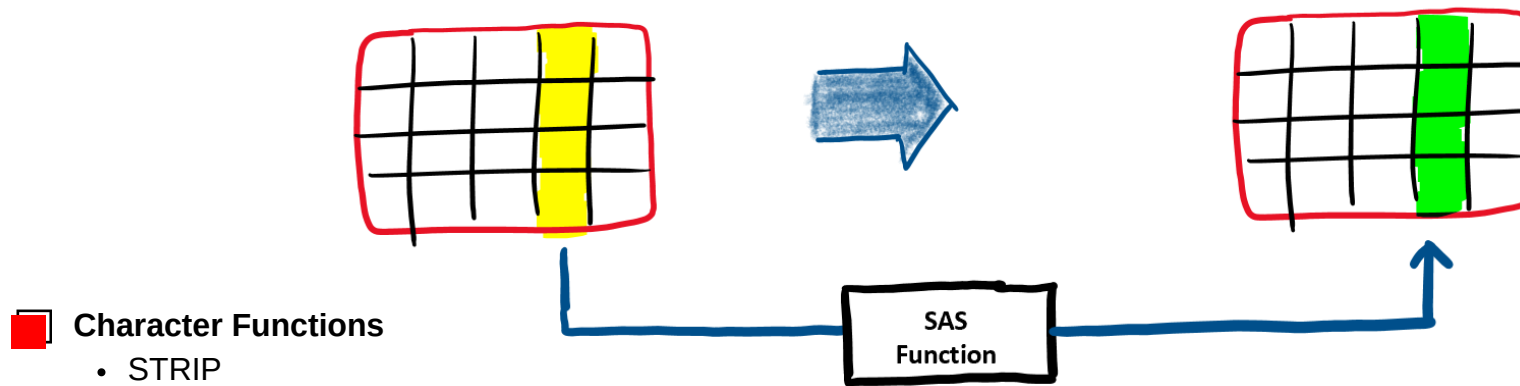
LEFT(string, n)

Where 'string' is the character string from which characters are to be extracted, and 'n' is the number of characters to extract.

The function returns the extracted characters as a new character string.

CHARACTER FUNCTIONS

STRIP



```
new_variable = STRIP(original_variable);
```

Here, new_variable is the name of the new variable that will contain the modified values from original_variable, and original_variable is the name of the original character variable that you want to modify.

The STRIP function in SAS is used to remove leading or trailing blanks or any specified characters from a character string. It takes a character string as an input and returns a modified character string with the leading or trailing blanks or specified characters removed. For example, if you have a character string ' Hello World ' and you apply the STRIP function to it, it will return 'Hello World'. If you specify a character to remove, such as '*', the function will also remove any occurrences of that character at the beginning or end of the string. The syntax for the STRIP function is: STRIP(string <, characters>).

Where:

- string: is the character string to modify
- characters: (optional) is a character or a list of characters to remove from the string. If not specified, blanks are removed.

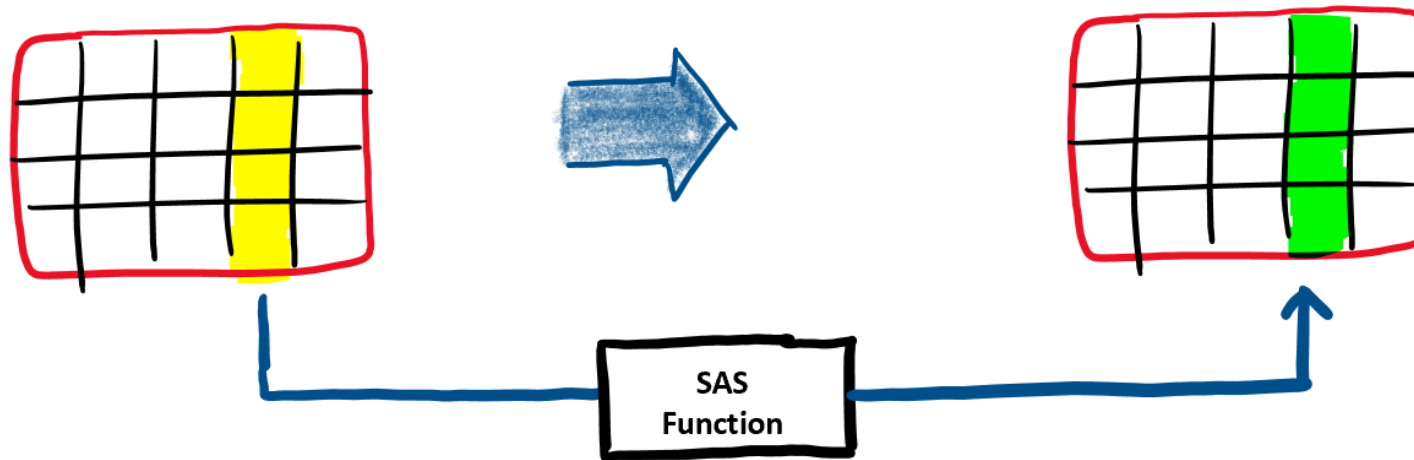
Here is an example of how to use the STRIP function:

```
data test; input string $; strip_string = strip(string); strip_star = strip(string, '*'); cards; ' Hello World ' 'Hello World' ; run;
```

In this example, the variable 'strip_string' will contain the value 'Hello World' and the variable 'strip_star' will contain the value 'Hello World'

CHARACTER FUNCTIONS

COMPRESS



Character Functions

- COMPRESS

COMPRESS(source, set)

Where "source" is the character string that you want to modify and "set" is a character string that specifies the characters that you want to remove from the source string. Default of "set" is empty/ leading/ trailing spaces.

The COMPRESS function in SAS is a character function that removes specified characters from a character string. It has the following syntax:

`COMPRESS(string, characters)`

The "string" argument is the character string from which you want to remove the specified characters. The "characters" argument is a string containing the characters that you want to remove.

For example, if you have a string "Hello World!" and you want to remove all exclamation points, you could use the following code:

```
result = COMPRESS("Hello World!", "!");
```

The result would be "Hello World".

You can also use the "characters" argument to specify a range of characters to remove. For example, to remove all vowels from a string, you could use the following code:

```
result = COMPRESS("Hello World!", "AEIOU");
```

The result would be "Hll Wrld!".

You can also use the "characters" argument to specify a list of characters to remove, like this:

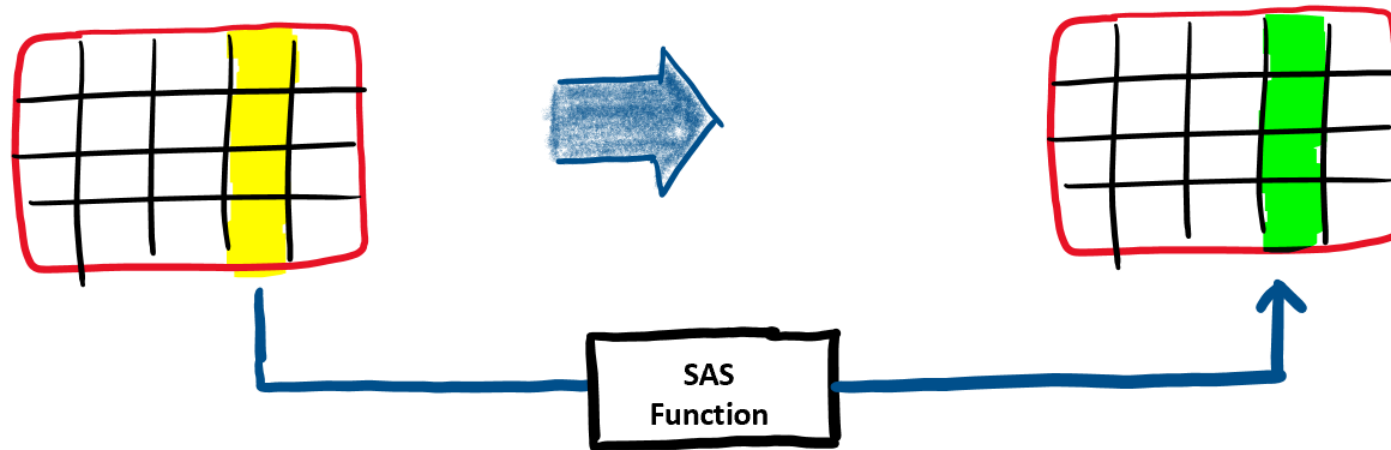
```
result = COMPRESS("Hello World!", "HWL");
```

The result would be "eo orld!".

The COMPRESS function is often used to clean up and standardize data, such as removing unwanted characters from strings before storing them in a database.

CHARACTER FUNCTIONS

COMPBL



Character Functions

- COMPBL

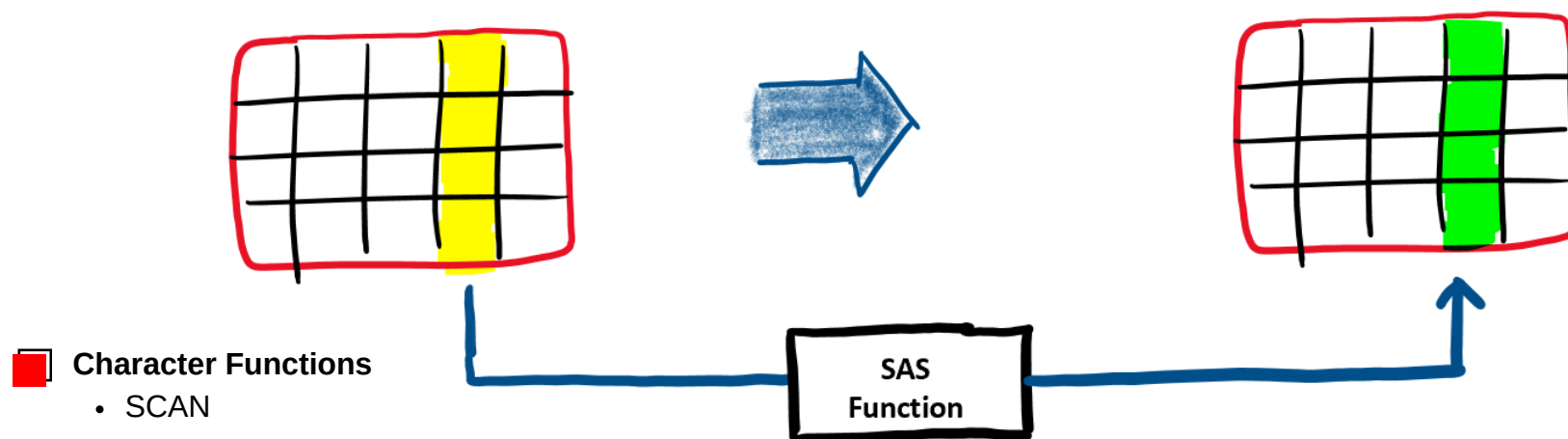
`compbl(string)`

The argument "string" is the character value that you want to compress and remove the blanks from. The function returns a character value with all the blanks removed from the original string.

The COMPBL function in SAS is a character function that removes consecutive spaces and reduces them to 1 space.

CHARACTER FUNCTIONS

SCAN



SCAN(string, n, delimiters) Where:

- string: The input character string that you want to extract substrings from.
- n: The position of the substring that you want to extract. For example, if n is 3, the function will extract the 3rd substring from the string.
- delimiters: A list of characters that are used to separate substrings in the input string.

The SCAN function in SAS is a character function that returns a specified number of substrings from a character string, based on a specified delimiter.

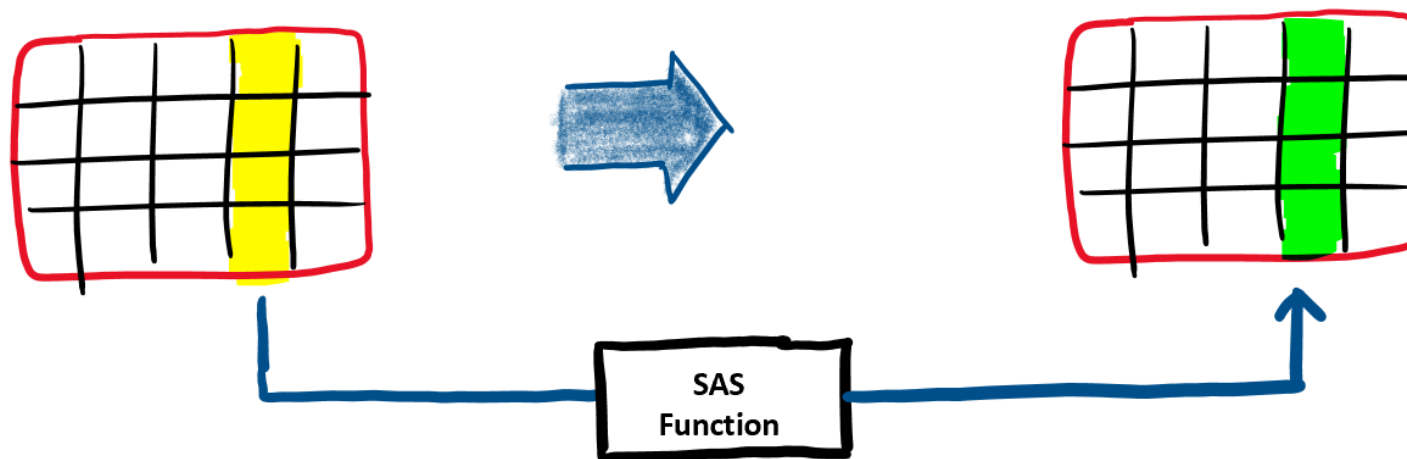
For example, if the input string is "United States of America" and the position argument is 2, the SCAN function will return the second substring from the input string, which is "States".

The delimiter in this case is a space character, and the position argument specifies which substring to return from the input string.

Here is an example of how you might use the SCAN function in SAS to extract the second substring from the input string "United States of America".

CHARACTER FUNCTIONS

INDEXC



Character Functions

- INDEXC

INDEXC(string, character or string to search for)

The INDEXC function in SAS is a character function that returns the position of a specific character or string within another character string. For example, if you want to find the position of the letter "a" in the string "United States of America", you can use the INDEXC function like this:

```
INDEXC("United States of America", "a")
```

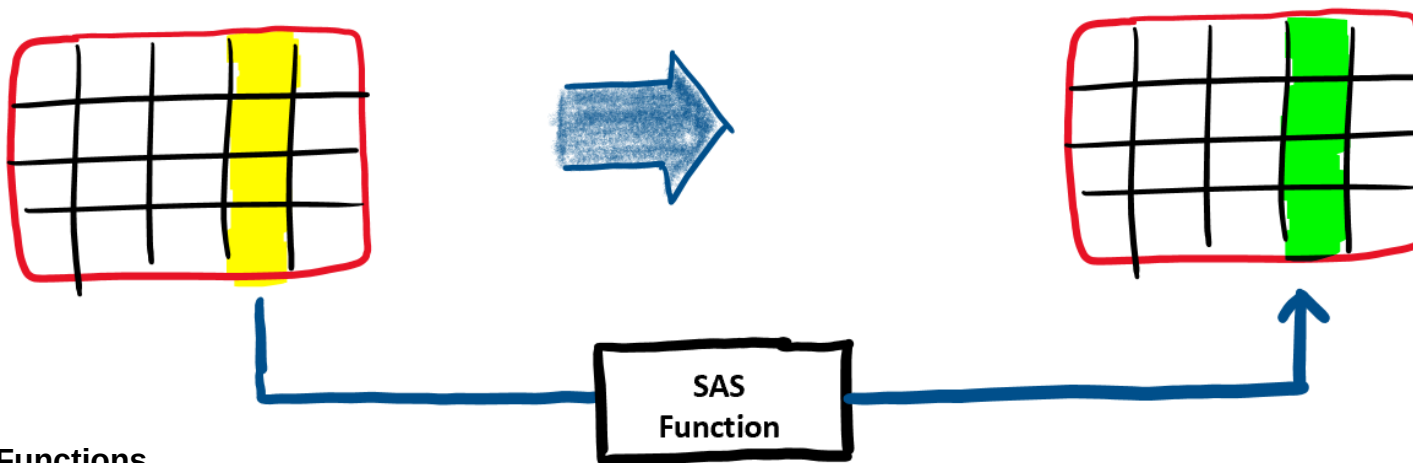
This would return the position of the first occurrence of the letter "a", which is 8. If you want to find the position of a different character or string, you can specify it as the second argument in the INDEXC function. For example, to find the position of the string "States" in the same string, you can use:

```
INDEXC("United States of America", "States")
```

This would return the position of the first occurrence of the string "States", which is 7. If the character or string you are searching for is not found in the string, the INDEXC function returns a value of 0.

CHARACTER FUNCTIONS

INDEXW



Character Functions

- INDEXW

INDEXW(string, character or string to search for)

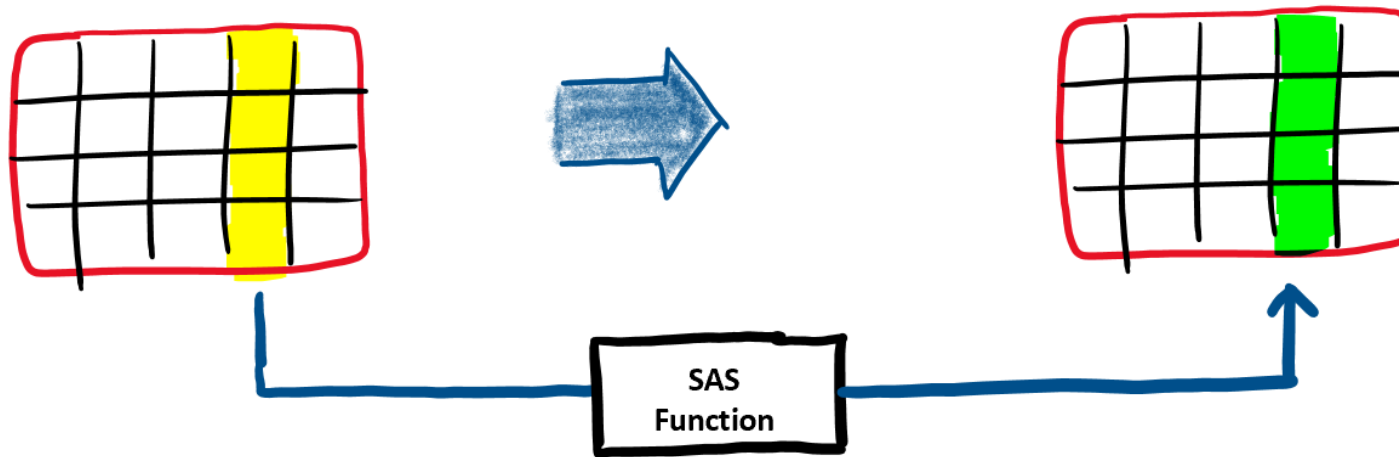
The INDEXW function in SAS is a character function that returns the position of the first occurrence of a word within a character string. It takes two arguments: the character string to search and the word to search for. The position is returned as a number, with the first word being at position 1. If the word is not found, the function returns a 0. For example:

```
data test; string = 'The quick brown fox jumps over the lazy dog'; pos = indexw(string, 'fox'); run;
```

In this example, the value of the variable 'pos' will be 6, because the word 'fox' is the 6th word in the string.

The INDEXW function is useful when you want to find the position of a specific word within a character string, rather than just the position of a specific character. It is particularly useful when working with text data, as it allows you to search for specific words rather than individual characters.

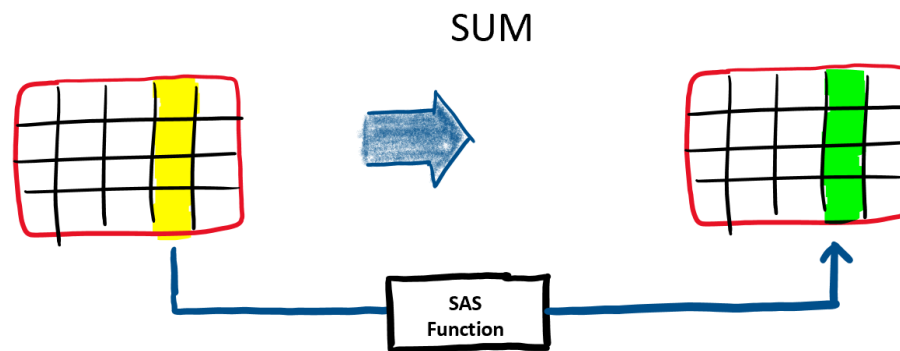
NUMERIC FUNCTIONS



Numeric Functions

- | | |
|---|---|
|  Sum |  Int |
|  Abs |  Min |
|  Ceil |  Max |
|  Floor | |

NUMERIC FUNCTIONS



Character Functions

- Sum

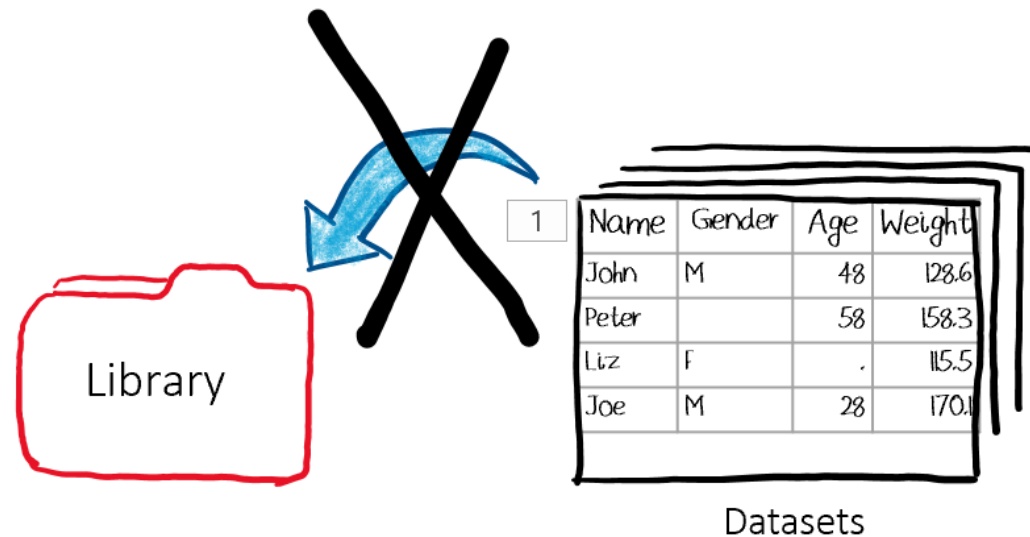
The SUM function calculates the sum of all the non-missing values in a numeric variable. Here is the basic syntax for using the SUM function:

SUM(expression)

The "expression" argument can be a single numeric variable, or it can be a combination of numeric variables and constants. The SUM function will return a numeric value that is the sum of all the non-missing values in the expression.

-
- In this example, the dataset contains two numeric variables: "salary" and "bonus".
 - The SUM function is used to add these two variables together to create a new variable called "netsal", which represents the total salary for each employee.
 - The SUM function ignores missing values when calculating the sum of a numeric expression.
 - This means that if an expression contains any missing values (denoted by a period), the SUM function will not include those values in the final sum.

NULL DATASETS



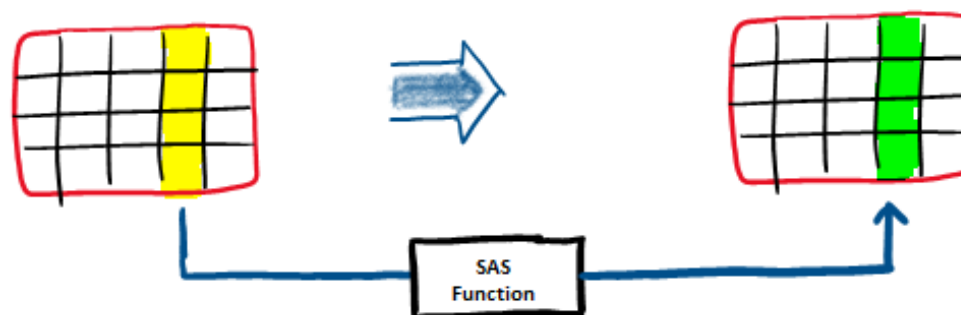
Null dataset

A null dataset in SAS is a special type of dataset that is not physically created or stored in the work library, but can still be used to create variables and perform calculations. The syntax for creating a null dataset is similar to that of a regular dataset, but the dataset name is “_NULL_”.

- A null dataset in SAS is a special type of dataset that is not physically created or stored in the work library, but can still be used to create variables and perform calculations.
- Unlike regular datasets, a null dataset does not have a physical presence and its contents are not accessible through the Explorer window.
- Instead, the contents of a null dataset can only be viewed using the PUT statement, which displays the values of variables in the log window.
- Null datasets are often used as a trick to perform calculations or demonstrate functions without creating and storing unnecessary datasets.

NULL DATASETS

ABC



Character Functions

- ABS

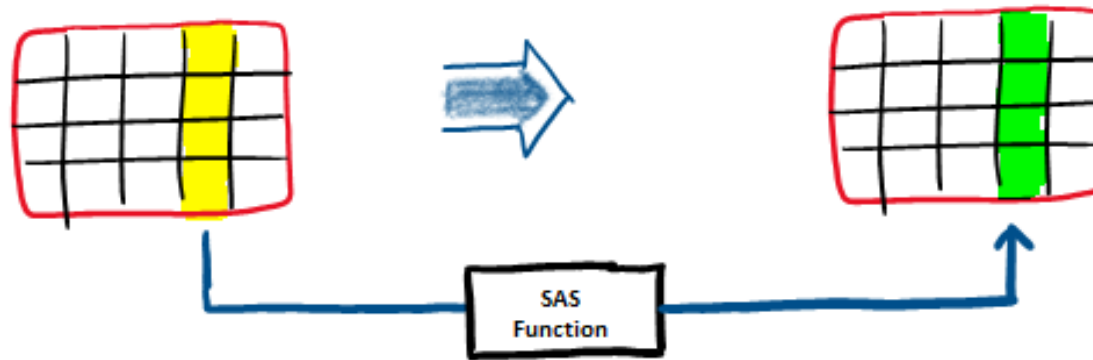
The ABS function in SAS is a function that returns the absolute value of a numeric variable.

ABS(expression)

The "expression" argument can be a single numeric variable, or it can be a combination of numeric variables and constants. The ABS function will return a numeric value that is the absolute value of the expression.

- A null dataset in SAS is a special type of dataset that is not physically created or stored in the work library, but can still be used to create variables and perform calculations.
- Unlike regular datasets, a null dataset does not have a physical presence and its contents are not accessible through the Explorer window.
- Instead, the contents of a null dataset can only be viewed using the PUT statement, which displays the values of variables in the log window.
- Null datasets are often used as a trick to perform calculations or demonstrate functions without creating and storing unnecessary datasets.

NUMERIC FUNCTIONS



Character Functions

- CEIL
- FLOOR
- INT

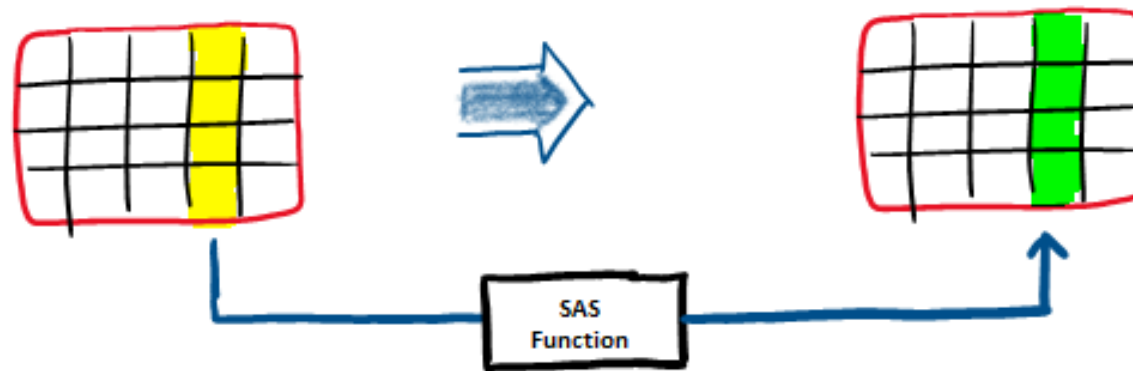
The CEIL, FLOOR, and INT functions in SAS are functions that return the smallest integer greater than or equal to a numeric expression, the largest integer less than or equal to a numeric expression, and the integer portion of a numeric expression, respectively. The CEIL function always returns the smallest integer that is greater than or equal to a numeric expression, while the FLOOR function always returns the largest integer that is less than or equal to a numeric expression. For example, if the input is 5.7, the CEIL function would return 6 and the FLOOR function would return 5.

The INT function simply returns the integer portion of a numeric expression, ignoring any decimal places. For example, if the input is 5.7, the INT function would return 5.

To use these functions in SAS, you simply specify the numeric expression within parentheses after the function keyword. For example:
`ceil_val = ceil(x); floor_val = floor(x); int_val = int(x);`

This would assign the smallest integer greater than or equal to the value of the variable "x" to a new variable called "ceil_val", the largest integer less than or equal to the value of the variable "x" to a new variable called "floor_val", and the integer portion of the value of the variable "x" to a new variable called "int_val".

NUMERIC FUNCTIONS



Character Functions

- MIN
- MAX

The SUM, MIN, and MAX functions in SAS are functions that return the sum, minimum, and maximum value, respectively, of a set of numeric variables. These functions ignore missing values when calculating the result, which means that if any of the variables in the set contain missing values, they will be excluded from the calculation.

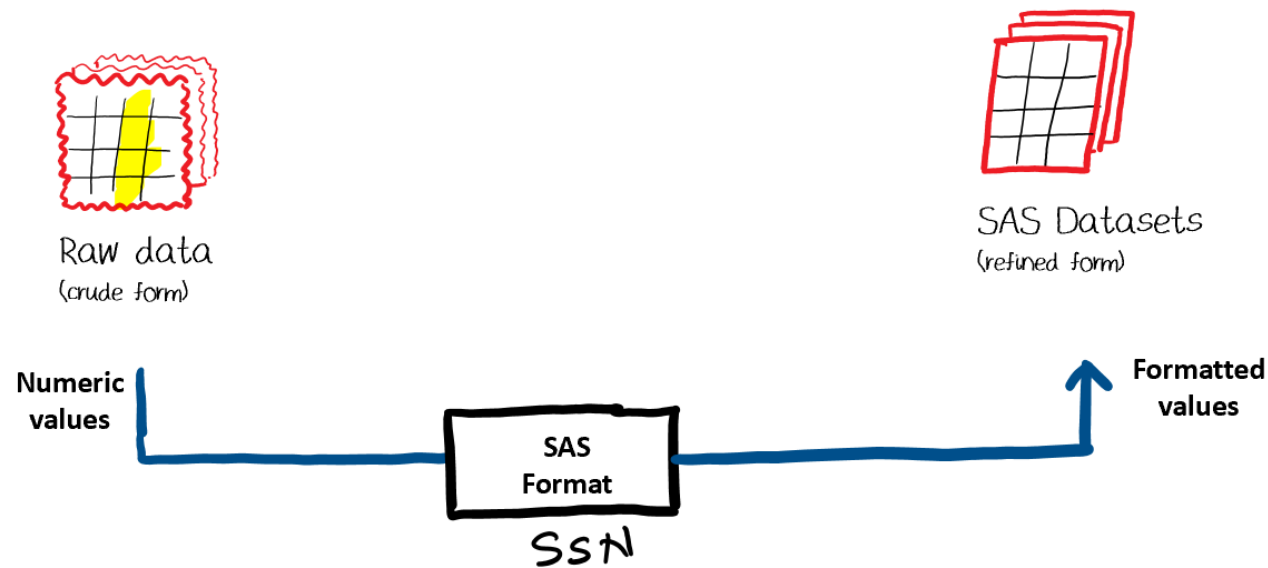
To use these functions in SAS, you simply specify the names of the variables within parentheses after the function keyword. For example:

```
sum_val = sum(x1, x2, x3); min_val = min(x1, x2, x3); max_val = max(x1, x2, x3);
```

This would assign the sum of the variables "x1", "x2", and "x3" to a new variable called "sum_val", the minimum value of the variables "x1", "x2", and "x3" to a new variable called "min_val", and the maximum value of the variables "x1", "x2", and "x3" to a new variable called "max_val".

You can use these functions in a data step to create new variables that contain the result of these operations applied to existing numeric variables. You can also use these functions in a proc step or in the SELECT clause of a proc SQL statement to perform calculations with the sum, minimum, or maximum value of a set of numeric variables.

SAS FORMATS

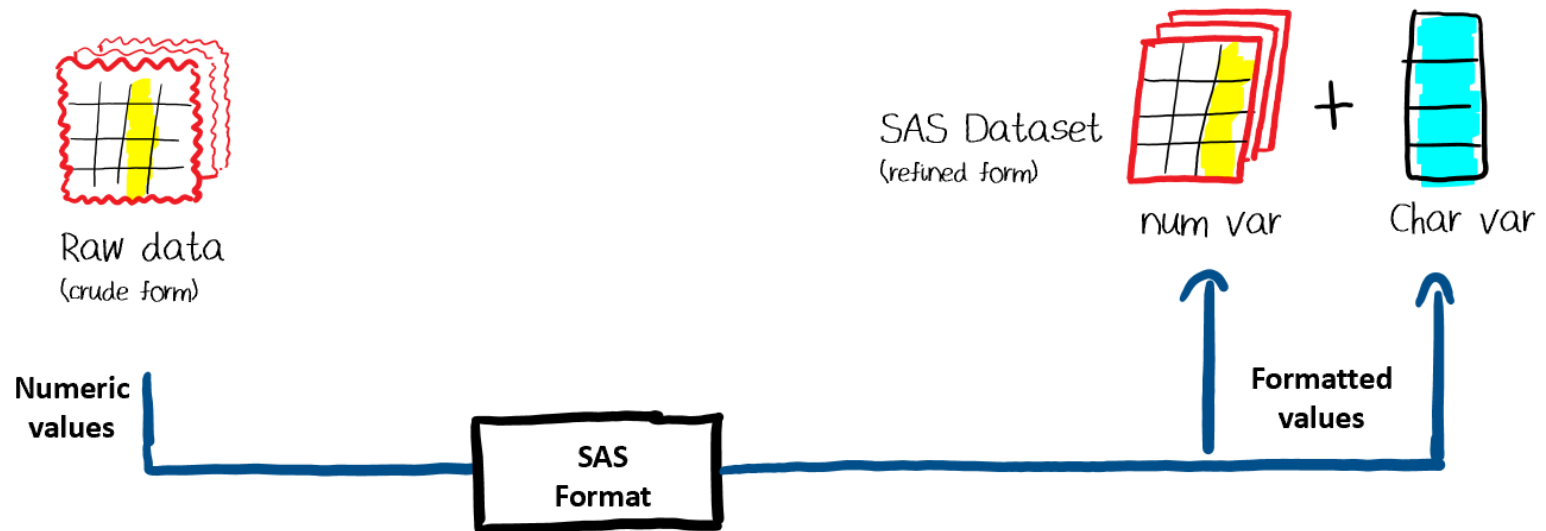


 **Formatting is making a numeric value appear more meaningful**

- SAS formats are rules or patterns that specify how to display data values in a certain way.
- They can be used to change the appearance of data values without changing the underlying data itself.
- For example, a SAS format might be used to display a numeric value as a dollar amount, with a dollar sign and a comma for thousand separators.
- SAS formats can be applied to both character and numeric variables.
- They are useful for making data values more meaningful and easier to read, especially when working with large datasets.
- There are many built-in SAS formats available, such as dollar formats, date and time formats, and formats for identifying values as missing or non-missing.
- You can also create custom SAS formats using format definitions.
- Formats are applied using the format statement or the put function in a SAS program.

SAS FORMATS

Two ways of formatting



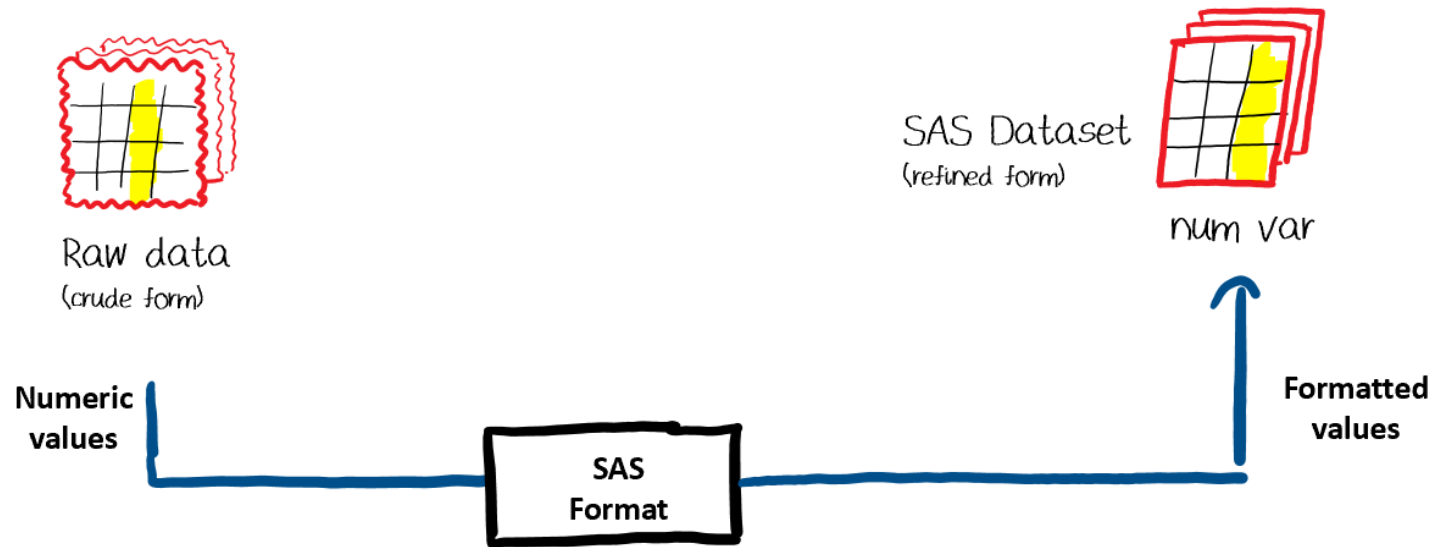
Two ways of formatting

- Format statement – remains numeric
- Put function – converts to char var

- Two methods of formatting data in SAS: using the format statement and using the put function.
- The format statement allows you to apply a SAS format to a numeric variable and display it in a certain way, while retaining the original raw values in the final dataset.
- The put function allows you to convert a numeric variable into a character variable, and apply a SAS format to it in the final dataset.
- Both methods result in the original raw values being displayed in a formatted way with additional characters, but the difference is in the type of variable that the final values are stored in.
- The format statement associates a particular format with a specific numeric variable, while the put function converts the numeric variable into a character variable and applies the format.

SAS FORMATS

Format statement



Two ways of formatting

- Format statement – original variable remains numeric
- `format salary dollar10.2 id ssn11.;`

The SAS format statement is a statement in a SAS program that associates a SAS format with a variable. It has the following syntax:
format variable-list <format-name>;

The variable-list is a list of one or more variables that you want to apply the format to. The format-name is the name of the SAS format that you want to use. The format-name can be either a built-in SAS format or a user-defined format.

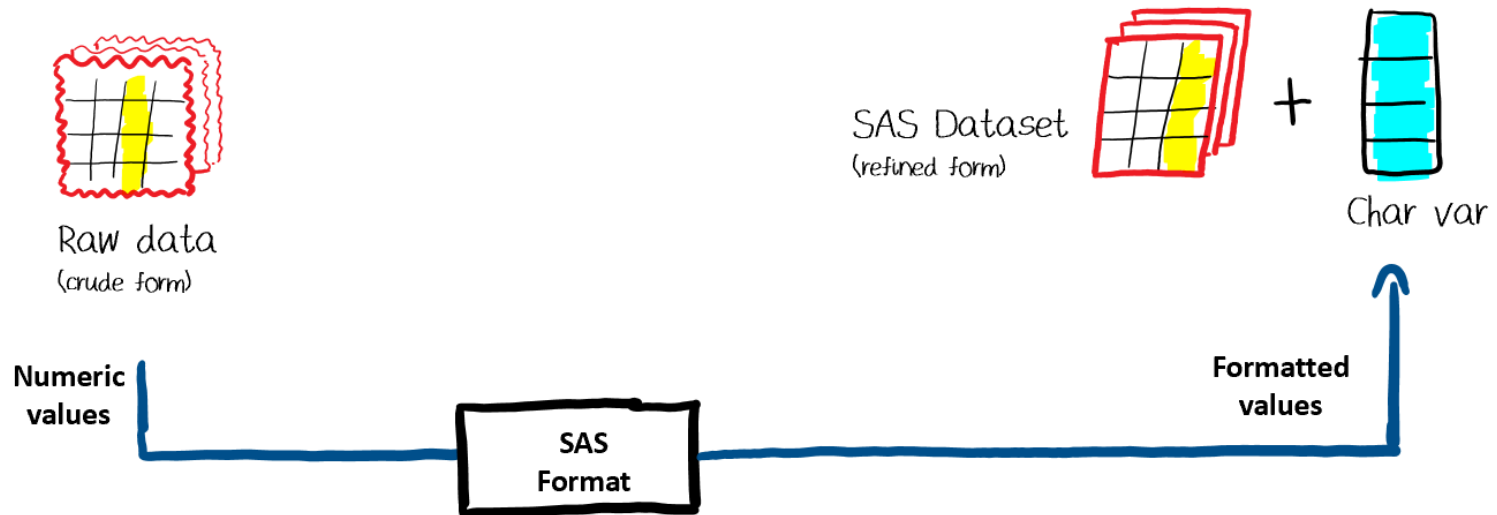
After the format statement is applied, the salary values will be displayed with a dollar sign and two decimal places, and the id values will be displayed as Social Security numbers. For example, the salary value 10000 will be displayed as \$10,000.00, and the id value 123-45-6789 will be displayed as is.

There are many different kinds of SAS formats, including:

- 1.** Numeric formats: These formats can be used to display numeric values in a specific way, such as with a certain number of decimal places or with a comma for thousand separators. For example, the dollar10.2 format displays a numeric value as a dollar amount with two decimal places, and the comma10.2 format displays a numeric value with a comma for thousand separators and two decimal places.
- 2.** Date and time formats: These formats can be used to display date and time values in a specific way. For example, the date9. format displays a date value as "mm/dd/yyyy", and the time5. format displays a time value as "hh:mm".
- 3.** Character formats: These formats can be used to display character values in a specific way. For example, the \$10. format displays a character value with a maximum length of 10 characters.

SAS FORMATS

Put function



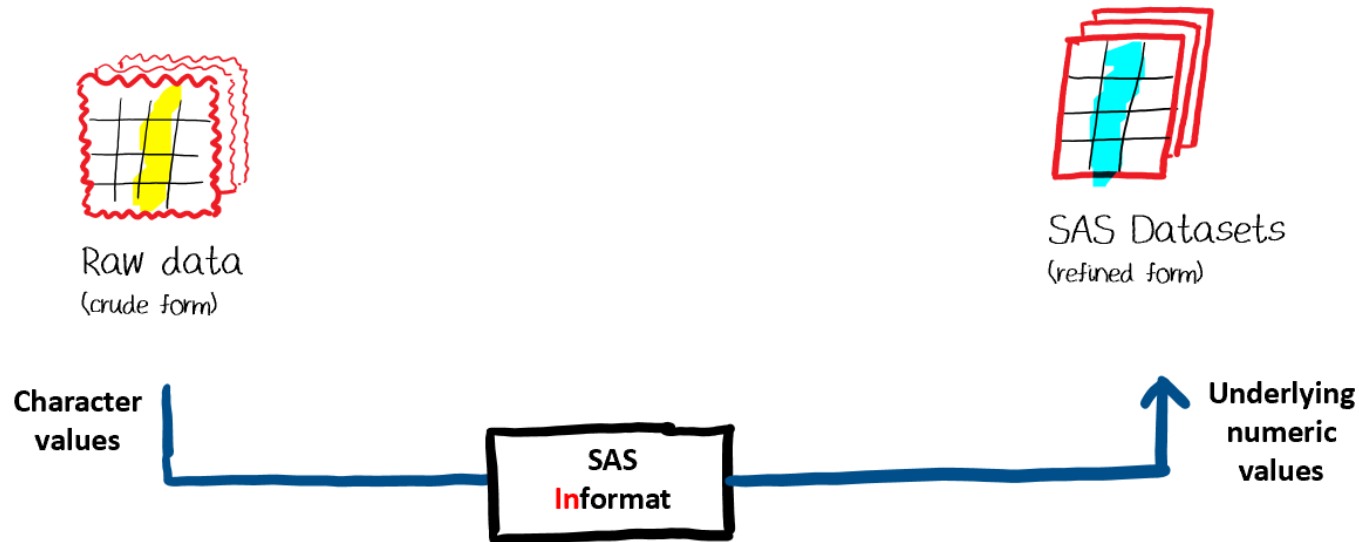
Two ways of formatting

- Put function – creates new char variable
`Salarytxt= put(salary,dollar10.2);`

The PUT function can be used to convert a numeric variable to a character variable and apply a format to it. The PUT function takes as input parameters the name of the variable to be formatted and the name of the format to be applied, and returns a character value that is the formatted version of the original numeric value.

In this code, the PUT function is used to apply the dollar10.2 format to the numeric variable salary, and create a new character variable called salarytxt that contains the formatted value. The dollar10.2 format displays the value with a dollar sign, a comma for thousand separators, and two decimal places.

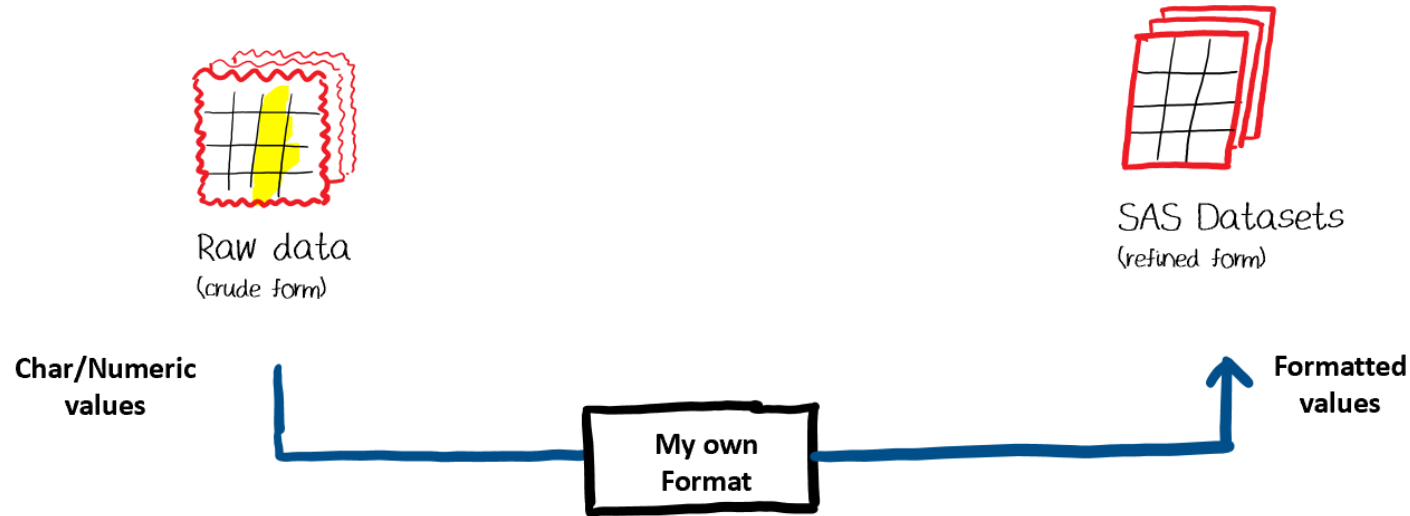
SAS INFORMATS



- InFormats work on pre-existing formats in raw data
- Results in underlying numeric values

- SAS informats are used to convert character data into numeric or date values.
- They are used in the opposite direction to formats, which take a numeric or date value and convert it into a character value with added formatting such as currency symbols or date separators.
- Informats are used in the INPUT statement to read data from external sources into SAS variables. They allow SAS to recognize and interpret character data in a specific way and convert it into a numeric or date value that can be used in SAS.
- The naming convention for SAS informats is similar to that of formats, where the informat name is followed by the X and Y parameters.
- The X parameter indicates the maximum length of the incoming character value, while the Y parameter indicates the number of decimal places in the value, if applicable.
- Some common SAS informats include MMDDYY., which recognizes and converts dates in the MMDDYY format, and DOLLAR., which recognizes and converts character data in the form of currency.

CUSTOM FORMATS



```
PROC Format;  
Value genfmt  
1='Male'  
2='Female';  
Run;
```

```
PROC Format;  
Value $genfmt  
'M'='Male'  
'F'='Female';  
Run;
```

-
- Custom formats in SAS allow you to create your own formats based on your specific needs for representing values in your data. This is done using the proc format procedure.
 - When creating custom formats, you must specify the name of the format, which must be from one to eight characters in length and cannot contain any numbers. You must also specify the values you wish to transform and how you want them to be represented in the final dataset. Custom formats can be applied to both numeric and character variables. After defining the custom formats, you can apply them to variables in a data set using the format statement.
 - Custom formats allow you to specify how you want the values of a variable to be displayed in your output or in reports.
 - For example, you can create a custom format to display the values 1 and 2 in a variable as "male" and "female" instead.
 - To create a custom format, you use the PROC FORMAT statement followed by a VALUE statement that specifies the values you want to format and the text or numeric values you want to display for those values

PROGRAM DATA VECTOR

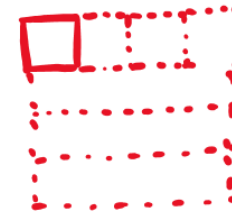


Program code

```
data class;  
    set sashelp.class;  
  
weightkg = weight * 0.454;  
  
run;
```



Somewhere inside your computers memory...



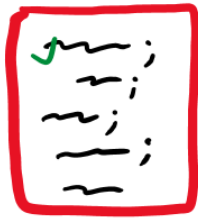
Observations {

Dataset Variables				Automatic Variables	
Name	Gender	Age	Weight	_N_	_ERROR_
John	M	48	128.6		
Peter		58	158.3		
Liz	F	.	115.5		
Joe	M	28	170.1		

Name	Gender	Age	Weight	_N_	_ERROR_
John	M	48	128.6	1	0
Peter		58	158.3	2	0
Liz	F	.	115.5	3	1
Joe	M	28	170.1	4	0

-
- The program data vector (PDV) is a temporary space in the computer's memory that is used by SAS to process and store data when executing a program.
 - SAS creates data sets in the PDV sequentially, starting with the creation of variables and then creating observations one below the other.
 - The PDV has a descriptive portion and a data portion. The descriptive portion contains information about the variables and their attributes, while the data portion contains the actual data values for each observation.
 - The PDV is used to store data temporarily during program execution and is cleared when the program is finished. It is important to understand the PDV in order to understand how SAS processes and stores data during program execution.
 - SAS creates data sets in the PDV sequentially, starting with the variables and then adding the observations one by one. The PDV consists of two parts: the Descriptor Portion (DP) and the Data Portion (DP). The DP contains information about the variables in the data set, such as their names and data types, while the DP contains the actual data values. The PDV also includes two automatic variables, the "_N_" variable which counts the number of observations in the data set, and the "_error_" variable which indicates if an error has occurred in a particular observation.

PROGRAM DATA VECTOR



Program code

```
data class;  
    set sashelp.class;  
  
weightkg = weight * 0.454;  
  
run;
```



Somewhere inside your computers memory...



Dataset Variables				Automatic Variables	
Name	Gender	Age	Weight	_N_	_ERROR_
John	M	48	128.6		
Peter		58	158.3		
Liz	F	.	115.5		
Joe	M	28	170.1		

Name	Gender	Age	Weight	_N_	_ERROR_
John	M	48	128.6	1	0
Peter		58	158.3	2	0
Liz	F	.	115.5	3	1
Joe	M	28	170.1	4	0

-
- Explicit output refers to the output that is explicitly requested by the user through code, such as creating a new data set or creating a table of summary statistics. This output is generated directly as a result of the code that is written by the user.
 - Implicit output, on the other hand, refers to output that is generated automatically by SAS as a result of running certain types of code, such as a PROC step or a DATA step.
 - Implicit output is not directly requested by the user, but is generated as a product of the code that is run. Examples of implicit output in SAS include log messages, notes, and warning messages.

PROGRAM DATA VECTOR

x	y
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

```
data myTable;
```

```
do i = 1 to 10;
```

```
  x=i;
```

```
  y=i;
```

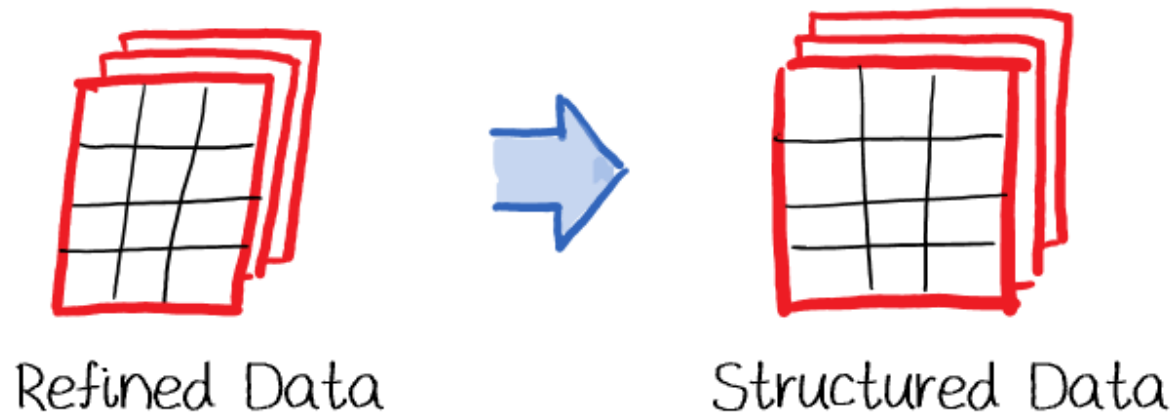
```
  output;
```

```
end;
```

```
run;
```

-
- A do loop is a type of loop that allows you to repeat a set of lines of code a specified number of times. It is useful when you want to iterate over a range of numbers or values.
 - The do loop is started with the "do" keyword, followed by a counter variable that is used to keep track of the number of iterations.
 - The loop is then ended with the "end" keyword. An explicit output statement can be used to create a separate observation for each iteration of the loop, and the loop can be modified by using the "by" keyword to specify the increment value.
 - In this code, the do loop starts with the keyword "do" followed by a counter variable, in this case "i", which is set to the starting value of 1.
 - The loop continues until the value of "i" is greater than 10. The loop increments "i" by 1 each time it is run.
 - Within the loop, the variables X and Y are both set to the current value of "i".
 - The "output" statement creates a new observation in the dataset for each iteration of the loop. The loop is ended with the "end" statement. Finally, the data step is run using the "run" statement.
 - After running this code, the resulting dataset "mytable" will contain 10 observations, with X and Y both taking on the values from 1 to 10.

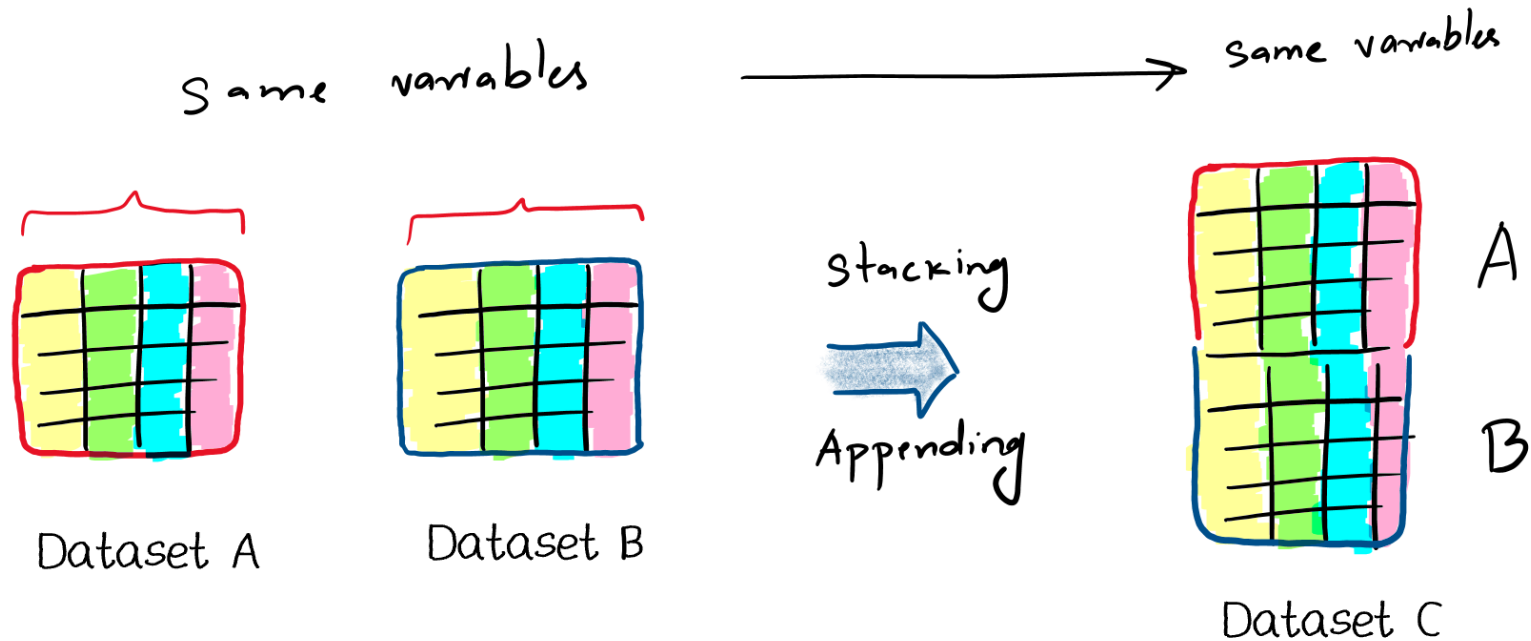
STRUCTURING DATA



- Stacking data
- Interleaving data
- Sorting data
- Removing duplicates

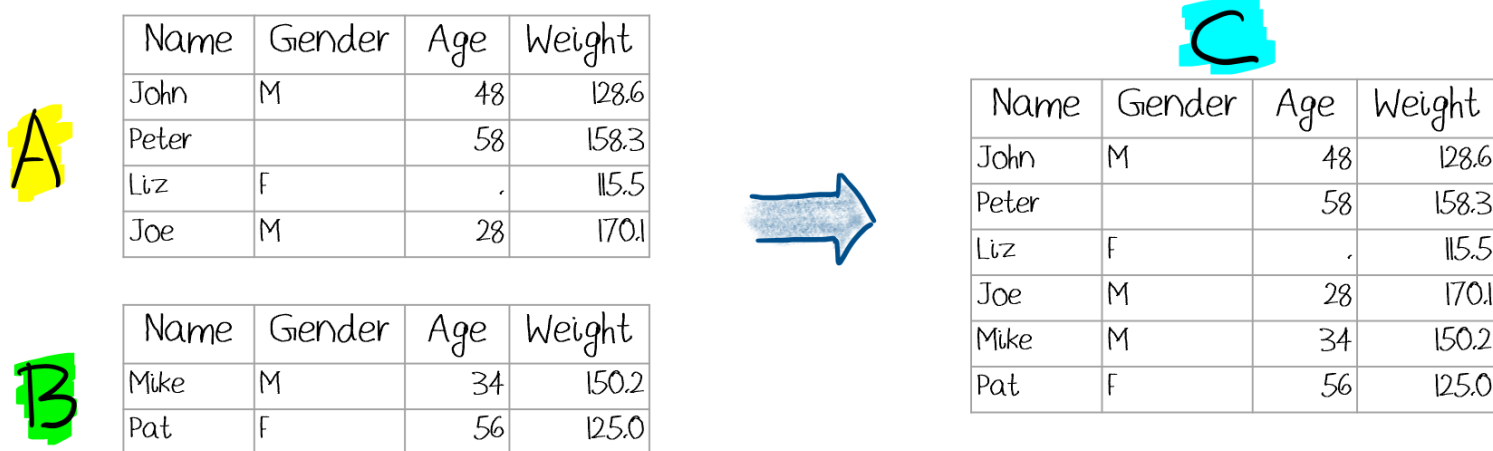
- Merging data and Joins
- Proc SQL
- Transposing data
- Retain statement

STACKING DATA



- Stacking data refers to combining two or more datasets together that contain the same set of variables.
- This can be done using a data step in SAS.
- In this example, there is a dataset A with the variables name, gender, age and weight, and a new dataset B with two additional observations for the same variables.
- The goal is to append B to A to create a combined dataset C with all observations from A and B.
- To accomplish this in SAS, the usual data step statement is used with the target dataset C and the set statement is extended to include A and B.
- It is important to note that the variable types in the datasets being stacked together should match.
- The example includes code that generates datasets A and B and then stacks them together to create dataset C.

STACKING DATA



```
data C;  
    set A B;  
run;
```

- Stacking data refers to pulling data from multiple datasets to create one unified dataset with all observations from individual datasets
- SAS has advanced procedures for stacking data in a more efficient way
- The simplest method of stacking data is through a data step, where you provide the individual dataset names and the name of the final dataset containing all observations

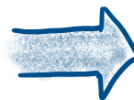
STACKING DATA

A

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

B

Name	Gender	Age	Weight
Mike	M	34	150.2
Pat	F	56	125.0



C

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1
Mike	M	34	150.2
Pat	F	56	125.0

```
proc append base= C data= A;  
run;  
proc append base=C data= B;  
run;
```

-
- The advanced method of stacking data is through a combination of multiple steps using the PROC APPEND procedure
 - The efficiency of the steps is measured in terms of the amount of time SAS takes to stack data, but in most cases, there is no difference in time
 - The decision of which method to use depends on the size of the datasets, with the data step being more suitable for small datasets and PROC APPEND being more suitable for large datasets with tens of thousands or millions of records
 - Two data sets, A and B, are used to demonstrate stacking using the proper procedure
 - The procedure used is to first stack data from A into C as the first step and then use another append to add observations from data set B into data set C in the second step
 - The syntax for proc append procedure includes the keywords "data" and "base", and the final target data set is specified in the "base equals to" statement
 - By the end of the step, all observations from data set A will have fed into data set C as the first step, and observations from data set B will be added in the second step
 - It is important that the number of variables and variable names in the appending datasets are exactly the same, otherwise issues may occur when using the procedure
 - One drawback of proc append is that it requires the exact same variables to be present, whereas the data step stacking does not
 - Proc append may be useful for stacking data sets with tens of thousands or millions of records for improved efficiency.

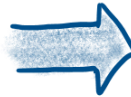
INTERLEAVING DATA

A

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

B

Name	Gender	Age	Weight
Mike	M	34	150.2
Pat	F	56	125.0



C

Name	Gender	Age	Weight
Peter		58	128.6
Liz	F	.	115.5
Pat	F	56	125.0
John	M	48	128.6
Joe	M	28	170.1
Mike	M	34	150.2

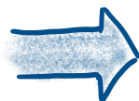
```
proc sort data=A;  
by gender;  
proc sort data=B;  
by gender;  
run;  
data C;  
set A B;  
by gender;  
run;
```

-
- Interleaving of data is a method of stacking data together where the final result is different from simple stacking
 - Interleaving rearranges observations based on certain criteria applied to a particular variable
 - The criteria is usually the ascending or descending order of the chosen variable
 - In interleaving, observations are grouped together based on values of the variable
 - The benefits of interleaving are to view groups of certain data together rather than being spread across the dataset
 - To interleave data, first sort or group the observations in the datasets and then stack the datasets together
 - The sorting or grouping is accomplished in the first step and the stacking is done in the second step
 - Interleaving is achieved by adding a statement in the data step which sorts the data by the variable being interleaved
 - The interleaving of data is demonstrated by sorting and grouping the observations in data sets A and B and stacking them together by the variable gender.

SORTING DATA

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

A



Name	Gender	Age	Weight
Joe	M	28	170.1
John	M	48	128.6
Liz	F	.	115.5
Peter		58	158.3

B

```
proc sort data=A out=B;  
by name;  
run;
```

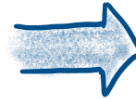
- Sorting data means rearranging the observations in a specified order
- For example, rearranging names in alphabetical order based on the variable name
- SAS has a powerful procedure called "proc sort" for sorting data
- Syntax: proc sort data=A out=B by name (can be ascending or descending by adding "ascending" or "descending" before the variable name)
- Default order of sorting in proc sort is ascending
- If a character variable with missing values is sorted, the missing values appear on the top by default
- If a numeric variable with missing values is sorted, they appear on the top by default

SORTING DATA

Multi -level

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F	.	115.5
Joe	M	28	170.1

A



Name	Gender	Age	Weight
Joe	M	28	170.1
John	M	48	128.6
Liz	F	.	115.5
Peter		58	158.3

B

```
proc sort data=A out=B;  
by gender age;  
run;
```

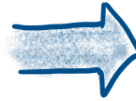
- Sorting can be done on one single level (variable) or multiple levels (variables)
- The order of sorting can be specified in the "by" statement
- The first level of sorting takes place first, and then subsequent levels of sorting follow
- SAS retains the original order of observations within the groups specified in the first level of sorting
- Proc sort can be used for multilevel sorting and filtering
- Filtering can be accomplished by using the "where" statement.
- The value of the filtering criteria must match the exact value in the data set

REMOVING DUPLICATES

NODUP option

A

Name	Gender	Age	Weight
John	M	48	128.6
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1



B

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1

- ❑ NODUP option – removes duplicates when every value in the all variables is exact duplicates

```
proc sort data=class out=sortclass nodup;  
by name ;  
run;
```

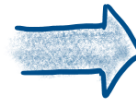
- Removing duplicates from within datasets is an important topic in SAS.
- This is done using the proc sort procedure.
- The proc sort procedure is used to sort and reorder observations within a dataset.
- The nodup option is used to remove duplicates from within a dataset.
- The nodup option works by removing duplicates that have exactly the same values in all variables.
- The syntax of the proc sort procedure starts with the keyword "proc", followed by "sort", followed by the input dataset, the output dataset and the nodup option.
- The by statement is used to specify the variable by which to sort.
- In the case discussed, the proc sort is applied to the class data set to remove the duplicates related to John.

REMOVE DUPLICATES IN DATA

NODUPKEY option

A

Name	Gender	Age	Weight
John	M	48	128.6
John	M	48	110.1
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1



B

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1

- ❑ NODUPKEY option – removes duplicates when every value in the all variables specified in by statement are exactly the same

```
proc sort data=class out=sortclass nodupkey;  
by name ;  
run;
```

- When some variables have different values, "no dupe" option cannot be used.
- SAS provides "no dupe key" option to handle such cases, where you can specify variables to check for duplicates.
- "no dupe key" removes duplicate observations, keeping the first one and removing the rest.
- To implement "no dupe key," use the "proc sort" statement followed by the input dataset and output dataset and include the "no key" option along with variables to check for duplicates.

MERGING DATA

Name	Gender	Age	Weight
John	M	48	128.6
Peter		58	158.3
Liz	F		115.5
Joe	M	28	170.1

A

Name	Height
John	165
Peter	155
Liz	176
Joe	152

B



Name	Gender	Age	Weight	Height
John	M	48	128.6	165
Peter		58	158.3	155
Liz	F		115.5	176
Joe	M	28	170.1	152

C

```
data C;  
merge A B;  
by name;  
run;
```

- Stacking of data refers to appending or putting data together, one below the other, with increased number of observations and same number of variables in the final data set.
- Merging of data refers to bringing together two or more data sets that contain a new set of data and new variables, with a common variable as the reference point for the merging.
- Merging of data is important when data from multiple sources need to be available in a unified data repository.
- The merging of data sets can be done using the "merge" statement in a data step in science, followed by the names of the data sets to be merged and the "by" statement for the reference variable.
- To properly run the code for merging, the data sets need to be sorted by the reference variable and matching "by" statements should be used in both the merging and sorting statements.

MERGING DATA

classweight

	Name	Gender	Age	Weight
	John	M	48	128.6
	Peter	M	58	158.3
	Liz	F	25	115.5
	Joe	M	28	170.1
	Pat	F	34	134.2
	Mike	M	34	201.1

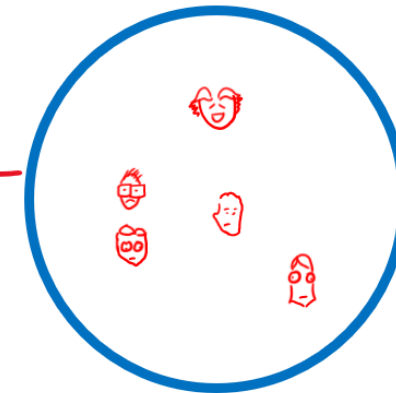
classheight

	Name	Height
	John	165
	Peter	155
	Liz	176
	Joe	152
	Tom	100.1

A



B



Joins

- ① Inner
- ② outer

-
- Merging data involves bringing data from multiple data sets and putting them together into one unified data set
 - This provides the ability to quickly obtain information from the data instead of having to search through multiple data sets
 - Joining data sets requires a common variable to connect them
 - Data sets A and B contain information about a class of six and five students respectively
 - The common variable between the two data sets is the name of the students
 - In terms of similarities, John, Peter, Liz, and Joe have records in both data sets A and B, allowing their height and weight to be determined
 - Differences include Pat and Mike only appearing in data set A, and Tom only appearing in data set B
 - The tabular structure of data set A and B is represented as a circle, denoting the data contained within the data set

MERGING DATA – INNER JOIN

classweight			
Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	25	115.5
Joe	M	28	170.1
Pat	F	34	134.2
Mike	M	34	201.1

A

classheight	
Name	Height
John	165
Peter	155
Liz	176
Joe	152
Tom	100.1

B

```
data joineddata;  
merge classweight(in=A) classheight(in=B);  
by name;  
if A and B;  
run;
```

- Inner join is a way to merge two data sets (A and B) to reveal the common observations between them.
- The result of an inner join between data sets A and B will contain the variables from both datasets (name, gender, age, weight, and height).
- Only the observations that belong to both data sets will be part of the inner join result.
- Inner join can be performed with multiple data sets as well.
- The syntax for performing an inner join includes a data step and a merge statement.
- The data step starts with the keyword "data" followed by the target data set.
- The merge statement lists the names of the data sets to be merged and specifies the common variable (name in this case) used to merge the datasets.
- The type of join performed is determined by the criteria specified in the if statement that follows the merge statement.
- The if statement starts with the keyword "if" and determines the observations that will be included in the inner join result.

MERGING DATA – FULL JOIN

	Name	Gender	Age	Weight
	John	M	48	128.6
	Peter	M	58	158.3
	Liz	F	25	115.5
	Joe	M	28	170.1
	Pat	F	34	134.2
	Mike	M	34	201.1

	Name	Height
	John	165
	Peter	155
	Liz	176
	Joe	152
	Tom	100.1

```
data joineddata;  
merge classweight(in=A) classheight(in=B);  
by name;  
if A or B;  
run;
```

-
- Full join is a type of join that combines all observations from two datasets, A and B.
 - The full join includes the intersection of datasets A and B and all observations from each dataset.
 - The full join in SAS can be performed using the data step with a statement using the keyword "or."
 - The resulting full join includes all the variables from both datasets, but may have missing data for observations that did not have data in the first place.

MERGING DATA – LEFT JOIN

	classweight			
	Name	Gender	Age	Weight
👤	John	M	48	128.6
👤	Peter	M	58	158.3
👤	Liz	F	25	115.5
👤	Joe	M	28	170.1
👤	Pat	F	34	134.2
👤	Mike	M	34	201.1

	classheight	
	Name	Height
👤	John	165
👤	Peter	155
👤	Liz	176
👤	Joe	152
👤	Tom	100.1

```
data joineddata;  
merge classweight(in=A) classheight(in=B);  
by name;  
if A;  
run;
```

- Left join in SAS merges datasets by including observations from the first (left) data set, regardless of whether they appear in the other data set or not.
- The determining factor for the observations in the final left join is the data set on the left, in this case data set A.
- The observations from the right data set (data set B) are merged as another column in the final data set, but those observations missing in data set B will result in missing values in the final left join.
- The left join is performed in SAS using a data step, followed by the name of the data set and the statement for data sets A and B, followed by the by statement and the "if a" statement to specify the type of join.
- Left joins are commonly used in real life settings as a reference data set to which multiple data sets can be combined into a unified data set for analysis or reporting.
- The left join is accomplished by writing "if a" statement in the data merge step.

MERGING DATA – RIGHT JOIN

	Name	Gender	Age	Weight
😊	John	M	48	128.6
😊	Peter	M	58	158.3
😊	Liz	F	25	115.5
😊	Joe	M	28	170.1
😊	Pat	F	34	134.2
😊	Mike	M	34	201.1

	Name	Height
😊	John	165
😊	Peter	155
😊	Liz	176
😊	Joe	152
😊	Tom	100.1

```
data joineddata;  
merge classweight(in=A) classheight(in=B);  
by name;  
if B ;  
run;
```

- Right join is the polar opposite of the left join.
- In a right join, the data from the right dataset (B) will appear in the final dataset, including the observations for John, Peter, Lewis, Joe, and Tom.
- The observations from Pat and Mike will not be part of the right join dataset because they do not belong to the region determined by B.
- The right join is accomplished using a data step and the "by" statement with the name variable, since it is the common variable between the two datasets and the determining factor.
- The final dataset will contain 5 observations, 4 of which are common between A and B and one is only from Tom.
- In the final right join dataset, the variables name and height will not have missing values because they come from B. However, the variables gender, age, and weight will have missing values for Tom.
- John, Peter, Liz, and Joe will have values for gender, age, and weight because they belong to both datasets A and B.

MERGING DATA – FAR LEFT AND RIGHT JOINS

	classweight					classheight	
	Name	Gender	Age	Weight		Name	Height
⚧	John	M	48	128.6	⚧	John	165
⚧	Peter	M	58	158.3	⚧	Peter	155
⚧	Liz	F	25	115.5	⚧	Liz	176
⚧	Joe	M	28	170.1	⚧	Joe	152
⚧	Pat	F	34	134.2	⚧	Tom	100.1
⚧	Mike	M	34	201.1			
	A					B	

```

data joineddata;
merge classweight(in=A) classheight(in=B);
by name;
if A and not B ;
run;

```

```

data joineddata;
merge classweight(in=A) classheight(in=B);
by name;
if B and not A;
run;

```

- Different types of joins: inner, full, left, right, far left, and far right join
- Far left join is a subset of left join that only includes observations that belong to data set A and not B
- Far right join is an extension of right join that only includes observations that belong to data set B and not A
- Inner join: common region between two data sets
- Full join: list of all observations from all datasets involved in the join
- Outer join: regions that belong to either left or right side of the merging, including far left and far right joins
- Final data set in far left join includes only observations from data set A (Pat and Mike)
- Final data set in far right join includes only observation from data set B (Tom)
- Missing values in final data sets are due to lack of corresponding observations in the other data set.
- Merging of data using SAS data step involves choosing the appropriate join to get the desired result
- The end result of merging data from multiple datasets is a single combined dataset for further reporting and analytics.

PROC SQL- SIMPLE DATA COPY

```
data B;  
    set A;  
run;
```

```
proc sql;  
create table B as  
select * from A;  
quit;
```

-
- Proc SQL is an advanced topic in science programming, an extension of the concepts discussed in SAS.
 - It has multiple purposes in science programming, including dataset modifications, creation of new variables, restructuring of data, and merging datasets.
 - Proc SQL is a structured query language (SQL) used in database programming.
 - SAS programmers can use Proc SQL to modify data sets, as it has tried to incorporate the power and elegance of SQL into SAS.
 - Proc SQL begins with the statement "proc sql;" and ends with a "quit statement;" instead of a "run statement".
 - Proc SQL can accomplish what other programming steps like the data step can do.
 - To copy data from one dataset to another in Proc SQL, one writes "create table B as select * from A;"
 - The number of statements involved in Proc SQL and the data step is pretty much the same.
 - In Proc SQL, to create a dataset, one says "create table (name of table)" followed by the necessary manipulation techniques.

PROC SQL – DATA FILTER

```
data B;  
    set A;  
where var = 'criteria';  
run;
```

```
proc sql;  
create table B as  
select * from A  
where var = 'criteria';  
quit;
```

-
- Proc SQL can be used to filter data
 - Syntax in proc SQL for filtering data is similar to that in a data step
 - In proc SQL, the first line is to indicate the target dataset, written as "create table B as"
 - The second line is to bring in all variables from the source dataset, written as "select * from a"
 - The third line is where the filtering condition is applied using the "where" statement
 - The filtering condition consists of the variable to be tested, followed by the condition statement such as "=", ">", "<", or "not equal to"
 - Double quotes are needed only if the variable is character in nature
 - The same data filtering can be accomplished in a proc SQL as in a data step
 - To apply a filter in a proc SQL, the where statement is added after the select statement
 - The semicolon must be the last character in the entire proc SQL statement.

PROC SQL- SORTING DATA

```
proc sort data=A out=B;  
by var1;  
run;
```

```
proc sql;  
create table B as  
select * from A  
order by var1  
;  
quit;
```

-
- Proc SQL can be used for sorting data in a similar way to Proc Sort.
 - In Proc SQL, you start with the "create table" statement to specify the output dataset.
 - The "as" keyword is used to specify the input dataset.
 - The "order by" statement is used to specify the variables to sort by.
 - Commas are used to separate multiple variables when sorting in Proc SQL.
 - The example uses the "Sacerdote" class dataset and sorts by the "sex" variable.
 - Then the "age" and "height" and "weight" variables are added to the sorting order.
 - The final output shows that the dataset is sorted by all the specified variables in the "order by" statement.

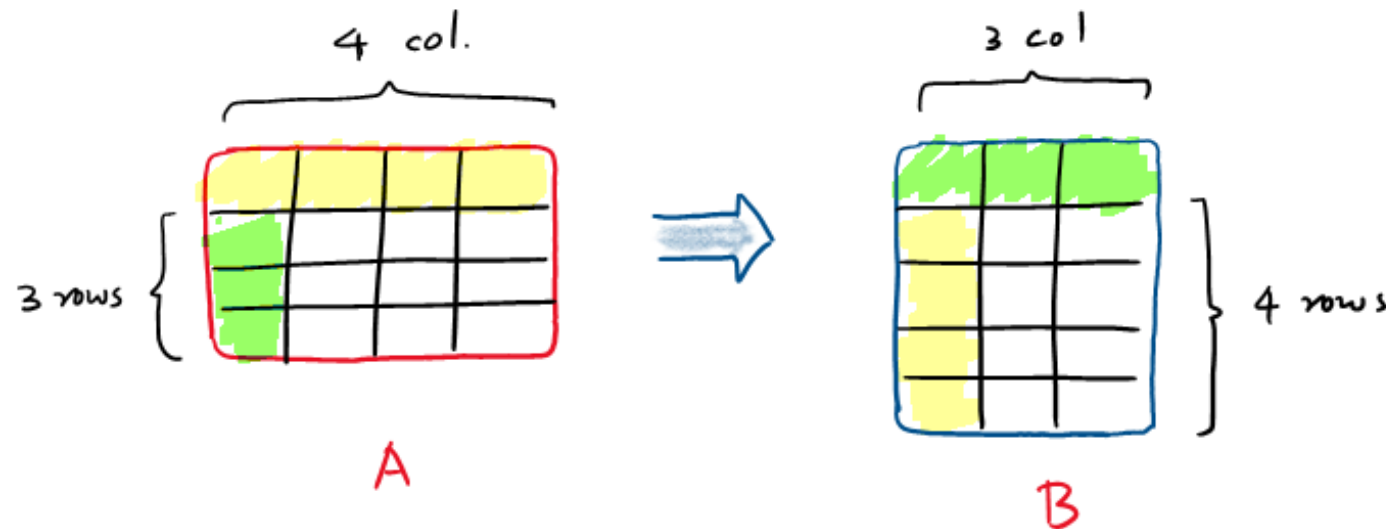
PROC SQL- REMOVING DUPLICATES

```
proc sort data=A out=B NODUP;  
by var1;  
run;
```

```
proc sql;  
create table B as  
select distinct * from A  
order by var1  
;  
quit;
```

-
- The next application of Proc SQL is removing duplicates within a dataset
 - The removal of duplicates is done by using the "nodup" option in the proc sort
 - The "nodup" option works by removing observations that are an exact match for every variable within the dataset
 - The same outcome can be accomplished in Proc SQL by introducing the "Distinct" keyword
 - The syntax of Proc SQL starts with "create table", followed by the output dataset, and then a select statement
 - The select statement uses the "distinct" keyword followed by an asterisk to compare every variable in the dataset for duplicates
 - The order by statement is optional in the select distinct statement and is used for sorting purposes

TRANSPOSING DATA



- Transposing of data in SAS means flipping over a data set.
- Variables in one data set become the observations in the transposed data set and observations become variables.
- Transposing of data is a unique feature in SAS that helps restructure data sets in a way not seen in other programming languages.
- Syntax for transposing of data in SAS:
 - Begins with "proc transpose"
 - Specify the variables to be transposed in the "var" statement
 - Identify the variable names in the final dataset with the "id" statement.
- Applications of transposing of data include converting data collected in a horizontal fashion into a single variable in the final dataset.

TRANSPOSING DATA

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1



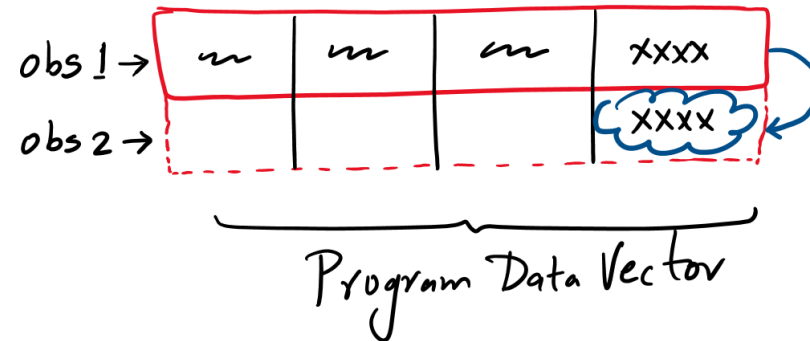
Transpose

Previous var	John	Peter	Liz	Joe
Gender	M	M	F	M
Age	48	58	45	28
Weight	128.6	158.3	115.5	170.1

```
proc transpose data=class;  
var gender age weight;  
id name;  
by sid name;  
run;
```

- The PROC TRANSPOSE in SAS is used to convert the columns of a SAS data set into rows. In the example you have provided, the input data set is named vitals, and the output data set is named t_vitals.
- The ID statement specifies the variable(s) that will become the new variables in the transposed data set. In this case, the ID variable is test. This means that the values of the test variable will become the new variables in the transposed data set.
- The BY statement specifies the variables that will be used to group the data. In this case, the BY statement variables are Sid and name. This means that the values of the vitals data set will be grouped by the Sid and name variables before they are transposed.
- The VAR statement specifies the variable(s) that will be transposed into the rows of the output data set. In this case, the VAR variable is value. This means that the values of the value variable will be transposed into the rows of the t_vitals data set.
- The PROC TRANSPOSE statement in this example will convert the columns of the vitals data set into rows in the t_vitals data set, with the values of the test variable becoming the new variables in the t_vitals data set, and the values of the value variable being transposed into the rows of the t_vitals data set. The Sid and name variables will be used to group the data.

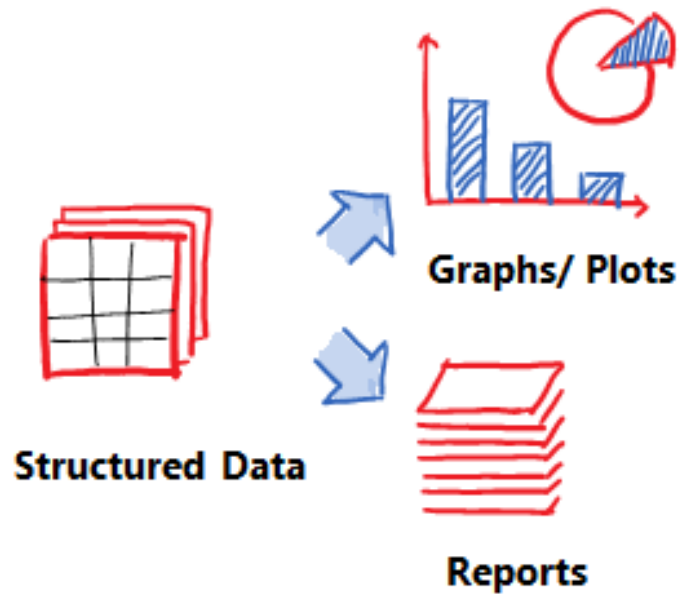
RETAIN STATEMENT



```
data salary1;  
  set salary;  
  by name month;  
  retain totalsal 0;  
  totalsal= totalsal + salary;  
  if first.name then totalsal=salary;  
run;
```

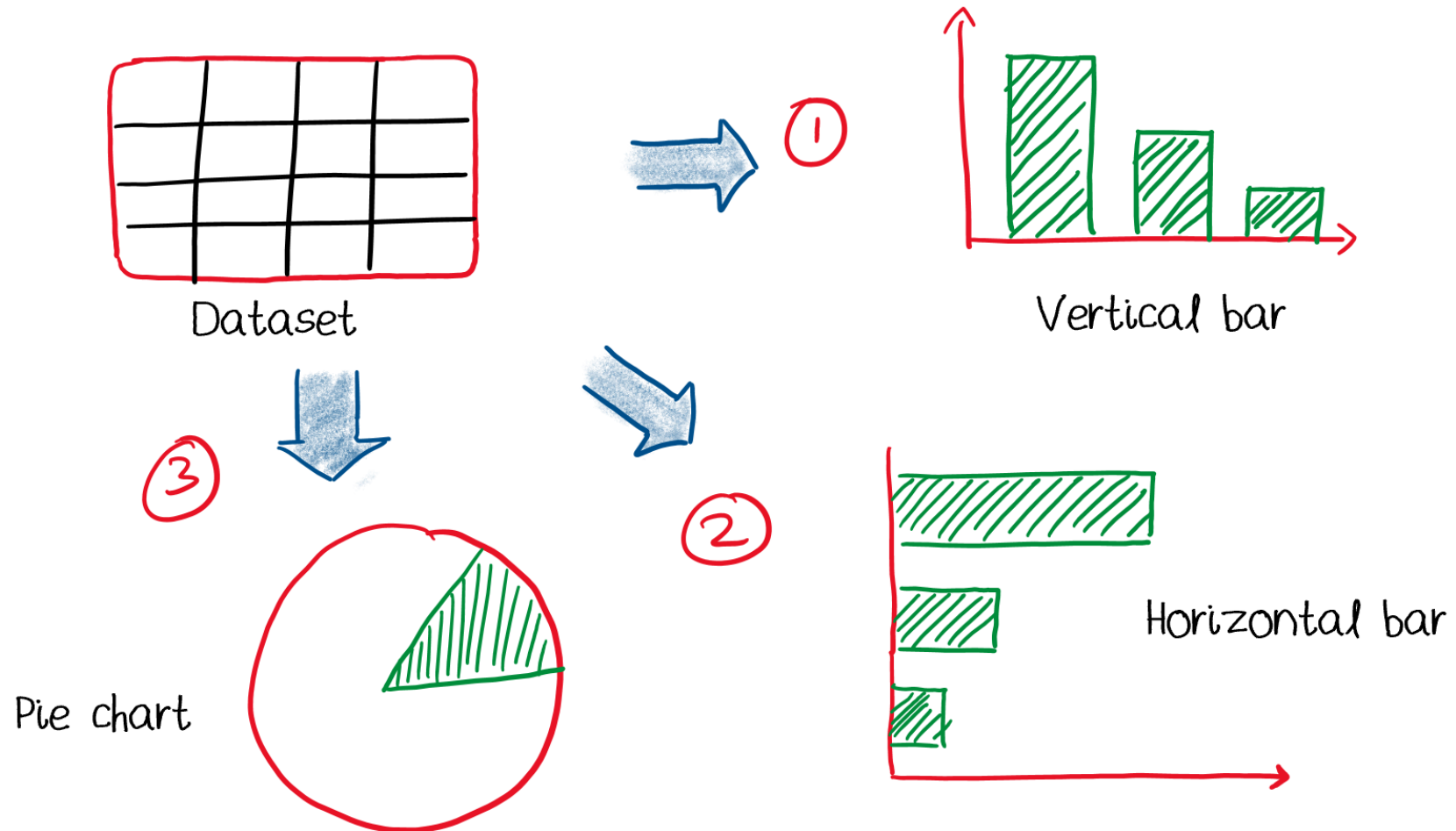
- The retain statement in SAS is used to retain or keep the value of a variable from one iteration to the next.
- It allows SAS to remember the value of a specific variable in the previous observation and use it as the initial value in the current observation. This is useful when creating cumulative variables or when the value of a variable needs to be carried forward to subsequent observations.
- The retain statement must be placed at the beginning of the SAS program and is applicable only within the context of a data step. It resets when a new observation is encountered or when the value of an identifying variable changes.
- The retain statement in SAS allows you to prefill a specific value of a variable in the subsequent observation of the dataset, even before the entire observation is complete or created.
- This process of retaining a particular value into the next observation is called a retain process, done through the retain statement.

SECTION 3: VISUALIZING DATA



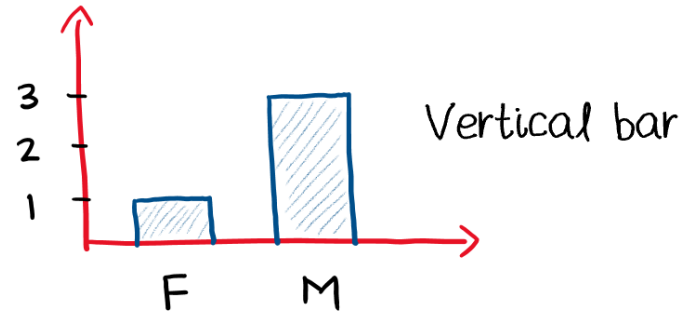
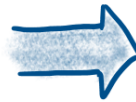
-  Charts
-  Plots
-  Proc Print
-  Statistical procs
-  Proc Report
-  Output Delivery System

CHARTS



CHARTS

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1

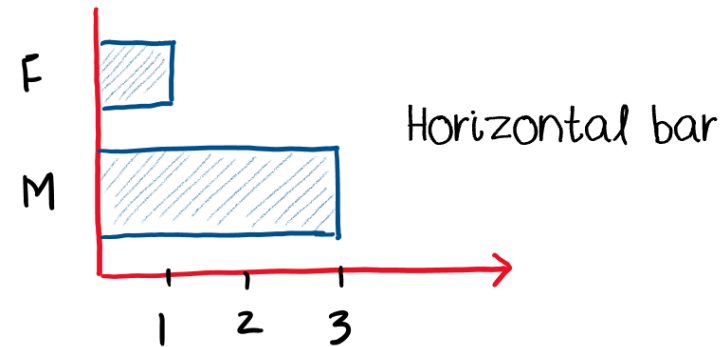
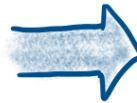


```
proc chart data=class;  
vbar gender;  
run;
```

- Proc Chart with VBAR is a SAS procedure used to generate a vertical bar chart from a structured dataset.
- It is a simple but powerful way to visualize the data quickly and efficiently, especially for large datasets.
- The statement includes the keyword "vbar" denoting vertical bar chart and the variable of interest specified.
- The output appears in the results window, with the Y-axis showing frequency count and the X-axis showing categories.
- This procedure is useful for gaining insight into the data and identifying trends and patterns.

CHARTS

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1

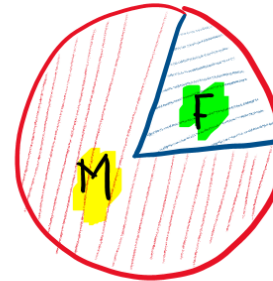
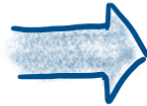


```
proc chart data=class;  
hbar gender;  
run;
```

- The Proc Chart with HBAR is a SAS procedure used to generate a horizontal bar chart from a structured dataset.
- It is similar to the vertical bar chart but displays the data horizontally, which may be useful in some cases.
- The statement includes the keyword "hbar" denoting horizontal bar chart and the variable of interest specified.
- The output appears in the results window, with the X-axis showing frequency count and the Y-axis showing categories.

CHARTS

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1



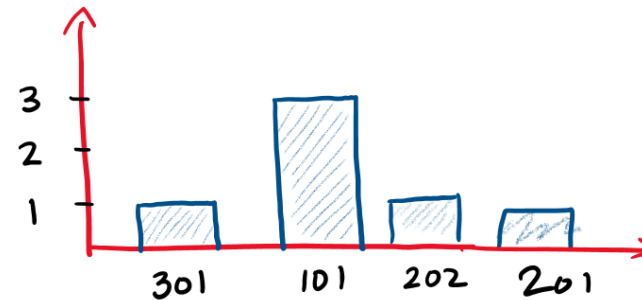
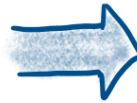
Pie chart

```
proc chart data=class;  
pie gender;  
run;
```

- The Proc Chart with PIE option is a SAS procedure used to generate a pie chart from a structured dataset.
- It is a circular chart divided into slices that represent different categories or proportions of a whole.
- The statement includes the keyword "pie" denoting pie chart and the variable of interest specified.
- The output appears in the results window, with the slices representing the categories and their proportions.
- This procedure is useful for comparing the proportions of different categories or variables and identifying patterns or relationships.

CHARTS

Name	Gender	Age	Course
John	M	48	101
Peter	M	58	101
Liz	F	45	201
Joe	M	28	301
Mike	M	34	202
Pat	F	23	201



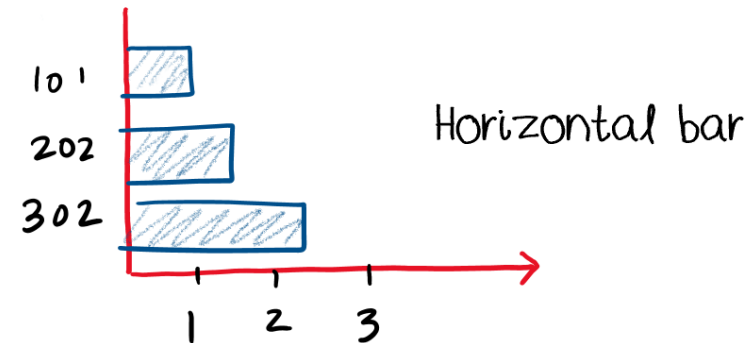
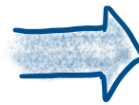
Vertical bar

```
proc chart data=class;  
vbar course/discrete;  
run;
```

- Proc Chart, the "vbar" statement creates a vertical bar chart.
- The "discrete" option used with the "vbar" statement is used when the variable of interest is categorical or nominal.
- When the "discrete" option is used, the chart will display all categories of the variable on the x-axis, even if some categories do not have observations.
- This is useful when you want to see all categories even if some categories have zero observations.

CHARTS

Name	Gender	Age	Course
John	M	48	101
Peter	M	58	101
Liz	F	45	201
Joe	M	28	301
Mike	M	34	202
Pat	F	23	201

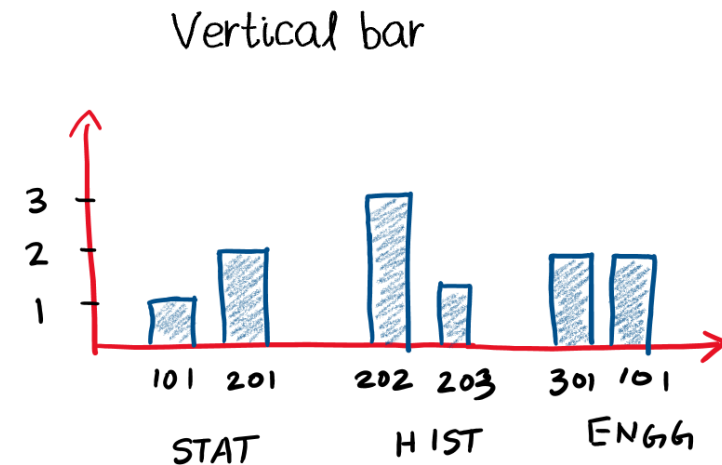


```
proc chart data=class;  
hbar course/ discrete;  
run;
```

- Similar to Proc Chart using vbar, the "hbar" statement creates a horizontal bar chart.
- The "discrete" option used with the "hbar" statement is used when the variable of interest is categorical or nominal

CHARTS

Name	Gender	Age	Course	Major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	45	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	23	201	STAT
Jul	F	45	101	HIST
Tom	M	56	301	ENGG



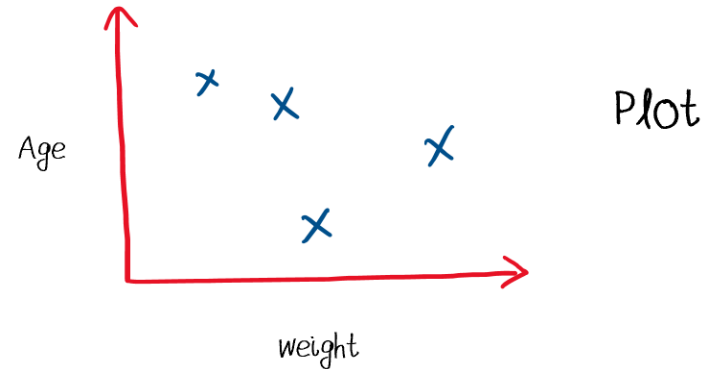
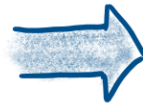
```
proc chart data=class;  
vbar course/discrete group=major subgroup=gender sumvar=age;  
run;
```

This SAS code is generating a vertical bar chart using the PROC CHART procedure with several options:

- The input dataset is named "class" and is specified after the PROC CHART statement.
- The VBAR statement is used to create a vertical bar chart.
- The variable of interest is "course" and is specified after the VBAR statement. The DISCRETE option is used to indicate that the variable is categorical.
- The GROUP option is used to create groups within the chart. The grouping variable is "major", which means that the chart will display separate bars for each value of "major".
- The SUBGROUP option is used to create subgroups within each group. The subgrouping variable is "gender", which means that the chart will display separate segments for each value of "gender" within each group of "major".
- The SUMVAR option is used to specify that the height of each bar will be the sum of the values of the "age" variable.

PLOTS

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1



```
proc plot data=class;  
plot age*weight;  
run;
```

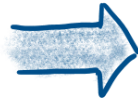
The SAS code `proc plot` produces a scatter plot of the age variable against the weight variable in the class dataset.

- `proc plot` is a procedure in SAS used to create various types of plots and graphs.
- `plot age*weight` is the plot statement, where age is plotted on the x-axis and weight on the y-axis.
- This plot can be used to visualize the relationship between the two variables and to identify any potential patterns or outliers in the data.

REPORT OUTPUT

Name	Gender	Age	Weight
John	M	48	128.6
Peter	M	58	158.3
Liz	F	45	115.5
Joe	M	28	170.1

Dataset



Obs	name	gender	age	weight
1	John	M	48	128.6
2	Peter	M	58	158.3
3	Liz	F	45	115.5
4	Joe	M	28	170.1

Report output

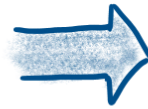
```
proc print data=class;  
var name gender age weight;  
run;
```

- Reports of data sets can be printed out in various formats using SAS's report output feature
- The process of generating report output is done through the proc print procedure
- Variables in the original data set and their values in the final report output contain the same data points
- The observation column generated in the report output gives the line number of the specific observation
- The var statement is used to list the variables to be included in the report output
- The verb clause statement can be added to filter observations to be included in the report output
- The proc print procedure is useful for viewing and printing out data sets in various formats.

REPORT OUTPUT

Name	Gender	Age	Course	Major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	45	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	23	201	STAT
Jil	F	45	101	HIST
Tom	M	56	301	ENGG

Dataset



name	gender	age	course	major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	25	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	34	201	STAT
Jil	F	45	101	HIST
Tom	M	56	301	ENGG

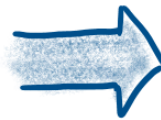
Report output

```
proc report data=class nowd;  
columns name gender age course major;  
define name/display;  
define gender/display;  
define age/display;  
define course/display;  
define major/display;  
run;
```

-
- Proc Report is a SAS procedure used to generate more complex and detailed report outputs compared to Proc Print.
 - It is capable of creating reports that are far more cosmetically appealing and can accomplish complex reporting.
 - Proc Report can list variables within a dataset, adjust column widths depending on contents, and provide further options that cannot be found in Proc Print.
 - Define statements follow the column statement for each variable that list options that can be associated with a specific column in the report.
 - Options such as display, order, group, analysis, across, and compute can be used in define statements to modify individual variables.
 - Proc Report can be used to write a proper report from the class dataset, which involves writing the column statement followed by the defined statement for each variable.

STATISTICAL PROCEDURES

Name	Gender	Age	Course	Major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	45	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	23	201	STAT
Jill	F	45	101	HIST
Tom	M	56	301	ENGG



The FREQ Procedure

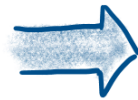
gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	3	37.50	3	37.50
M	5	62.50	8	100.00

```
proc freq data= class;  
tables gender;  
run;
```

-
- Statistical procedures can be used to perform quick calculations on a dataset and obtain reports that provide more insight into the data.
 - The proc freq procedure is used to generate frequency reports for character variables that are categorical.
 - The output of the proc freq procedure is a report that contains the variable broken down into individual categories followed by the frequency column that lists the number of occurrences of each category.
 - The report also includes the percentage column that gives the percentage of each category and the cumulative frequencies that give the ongoing frequency of each category.
 - To use the proc freq procedure, the data set is specified using the data statement, followed by the tables statement that specifies the variable to obtain the frequency counts or the summary report, followed by the name of the variable.
 - The output result of the proc freq procedure is displayed in the results output rather than in the output data tab, as it is a summary report.
 - The proc freq procedure takes missing data into account in calculating percentage counts, and it evaluates the percentage counts based on the total number of non-missing values in the denominator.

STATISTICAL PROCEDURES

Name	Gender	Age	Course	Major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	45	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	23	201	STAT
Jill	F	45	101	HIST
Tom	M	56	301	ENGG



The MEANS Procedure

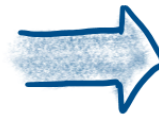
Analysis Variable : age				
N	Mean	Std Dev	Minimum	Maximum
8	41.0000000	12.5470543	25.0000000	58.0000000

```
proc means data=class;  
var age;  
  
run;
```

- The proc means procedure is used to obtain summary reports or simple statistics for numeric variables.
- The output of the proc means procedure includes information about the number of observations, mean value, standard deviation, minimum value, and maximum value of the analysis variable.
- The Proc Means procedure works best with numeric analysis variables because it can obtain various statistical parameters meaningfully.
- The output of the proc means procedure for the "age" variable in the "class" data set shows the number of observations, mean age, standard deviation, minimum age, and maximum age of students.

STATISTICAL PROCEDURES

Name	Gender	Age	Course	Major
John	M	48	101	STAT
Peter	M	58	101	HIST
Liz	F	45	201	STAT
Joe	M	28	301	ENGG
Mike	M	34	202	HIST
Pat	F	23	201	STAT
Jill	F	45	101	HIST
Tom	M	56	301	ENGG



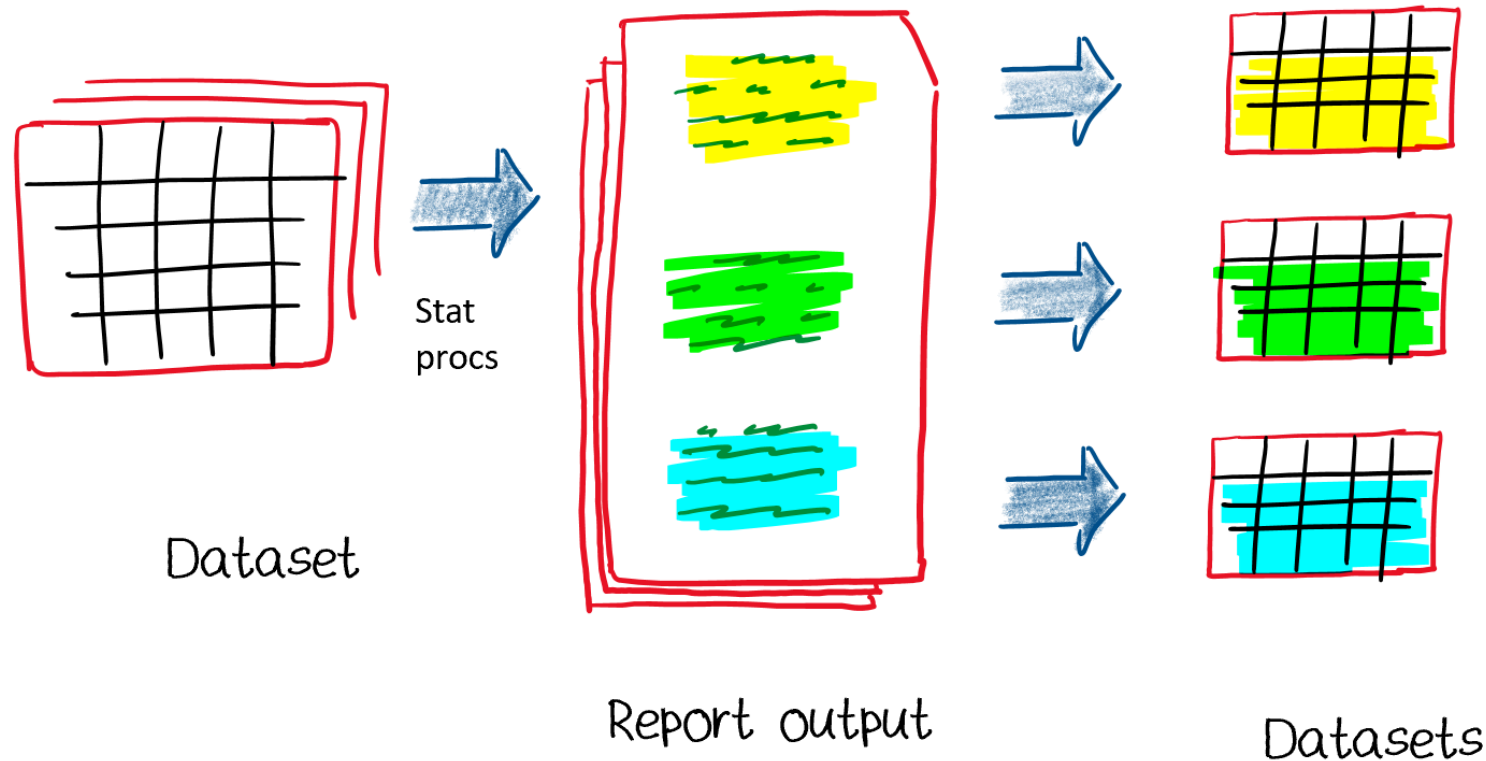
The UNIVARIATE Procedure			
Variable: age			
Moments			
N	8	Sum Weights	8
Mean	41	Sum Observations	328
Std Deviation	12.5470543	Variance	157.428571
Skewness	0.1654751	Kurtosis	-1.6393111
Uncorrected SS	14550	Corrected SS	1102
Coeff Variation	30.6025714	Std Error Mean	4.43605359
Basic Statistical Measures			
Location		Variability	
Mean	41.00000	Std Deviation	12.54705
Median	39.50000	Variance	157.42857
Mode	34.00000	Range	33.00000
		Interquartile Range	21.00000
Tests for Location: Mu0=0			
Test	Statistic	p Value	
Student's t	t 9.242449	Pr > t	<.0001
Sign	M 4	Pr >= M	0.0078
Signed Rank	S 18	Pr >= S	0.0078

```
proc univariate data=class;
var age;

run;
```

- Proc univariate is another statistical procedure that can be used to obtain detailed statistics for numeric variables.
- It provides more complex and detailed statistical parameters in addition to the basic ones obtained from proc means.
- The syntax for univariate is very similar to proc means, except for the change in the name of the procedure.
- The output from univariate consists of multiple sections of reports that can be used based on statistical needs.

OUTPUT DELIVERY SYSTEM (ODS)



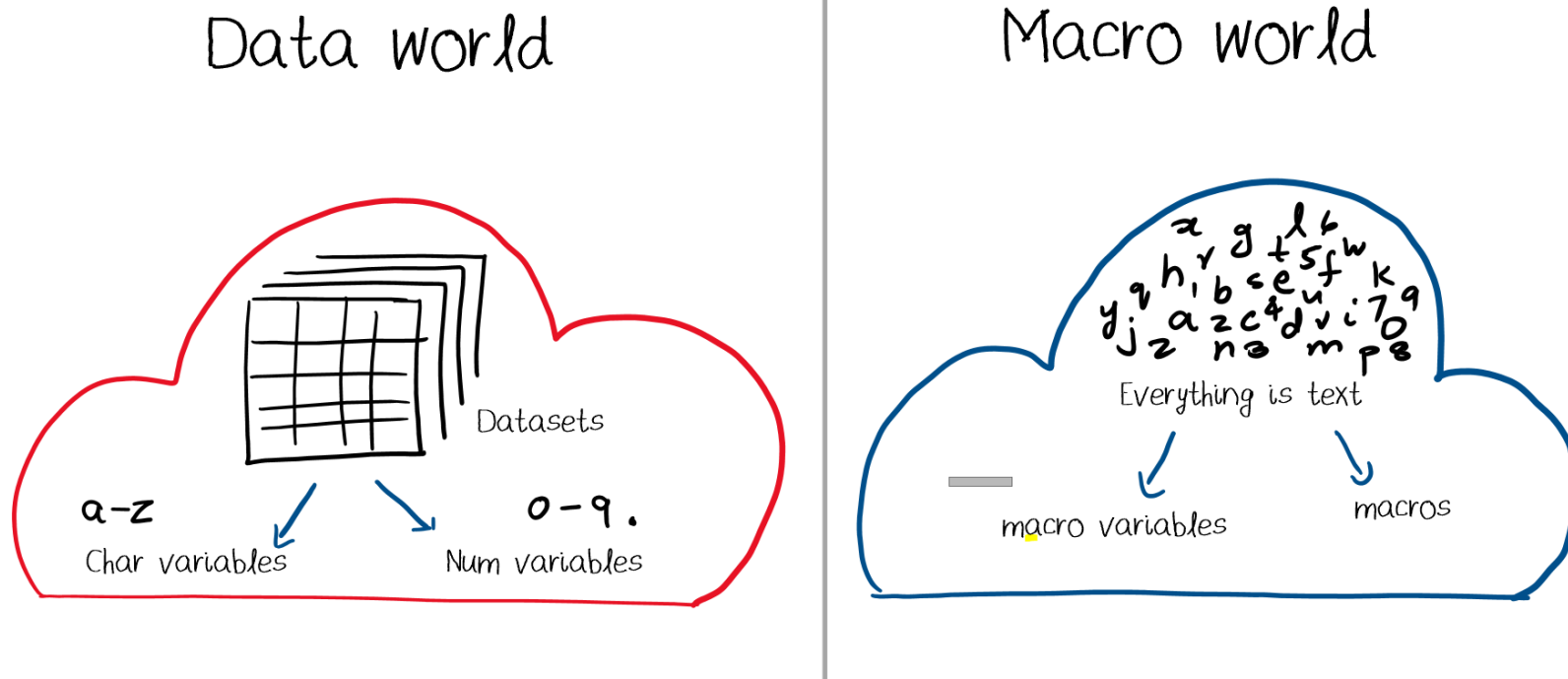
- SAS provides a powerful capability called the Output Delivery System (ODS) to customize the appearance of output for final deliverables.
- The ODS can be used for many applications, such as creating output in different file formats, embedding graphics and colors, and creating presentations.
- There are several formats available in SAS, including PDF, HTML, and RDF.
- A dataset can be used with the PROC PRINT procedure to obtain the default report output in SAS.
- Different formats can be downloaded and viewed in their respective applications, such as Microsoft Word or a web browser.

OPTIMIZING CODE



- Macro variables
- Ampersand resolutions
- SAS Macros
- Macro functions

SAS MACROS



- Macros are used to optimize code and make it more efficient by removing repetitions.
- SAS can be divided into two worlds: the data world and the macro world.
- The data world deals with variables and observations within data sets, while the macro world treats everything as text.
- In the macro world, there are no data sets or numeric/character variables, only macro variables.

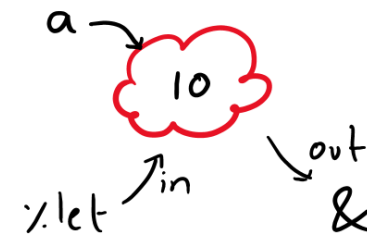
MACRO VARIABLES

Data world

a
10

```
data test;  
a=10; ← Data variable  
run;
```

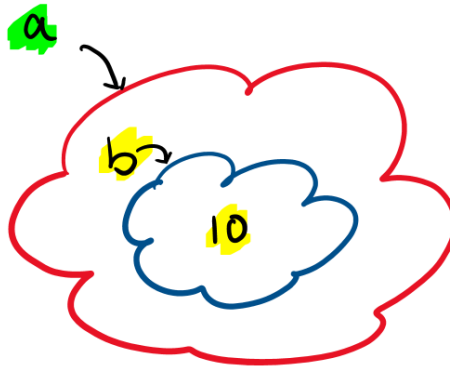
Macro world



```
%let a= 10; ← Macro variable  
%put -----> &a;
```

- Macro variables are created using the %let keyword and are assigned values using the = symbol.
- Everything in the macro world is treated as text, and macro variables can be assigned either numeric or character values.

NESTED MACRO VARIABLES



```
%let b=10;  
%let a=b;  
  
%put ----> &&a;
```

-
- Nested macro variables contain a macro variable within another macro variable.
 - To retrieve the values within the nested macro variables, consecutive ampersands are used for retrieving nested values.

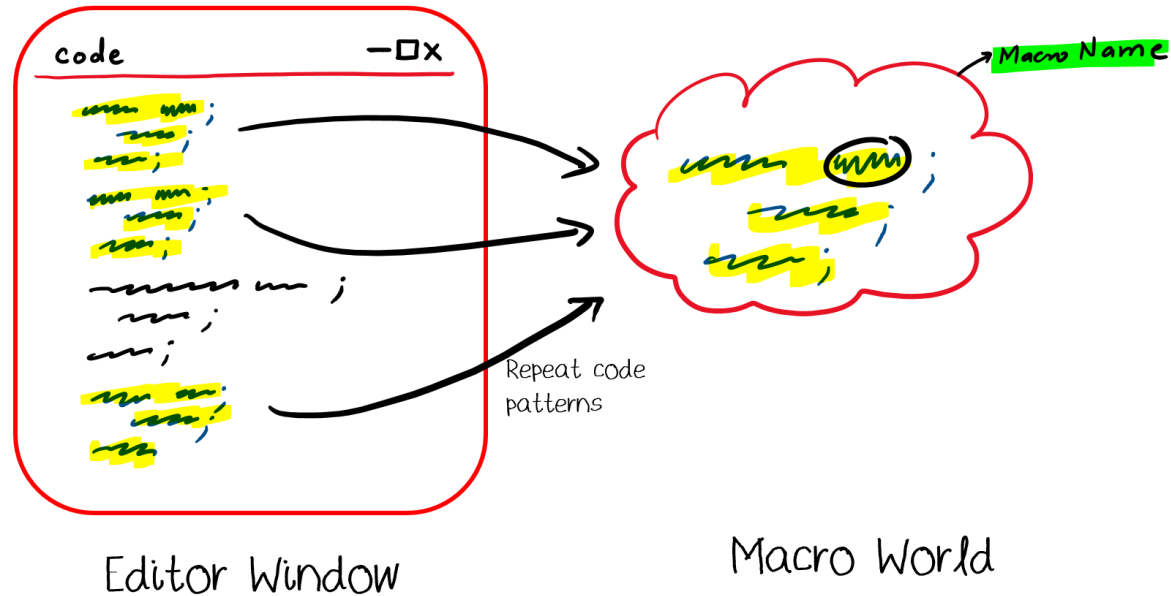
RULES OF & RESOLUTION

1. Left to right
2. All consecutive double &s are resolved into a single &
3. Till all &s are resolved

```
%let b=10;  
%let a=b;  
  
%put ----> &&&a;
```

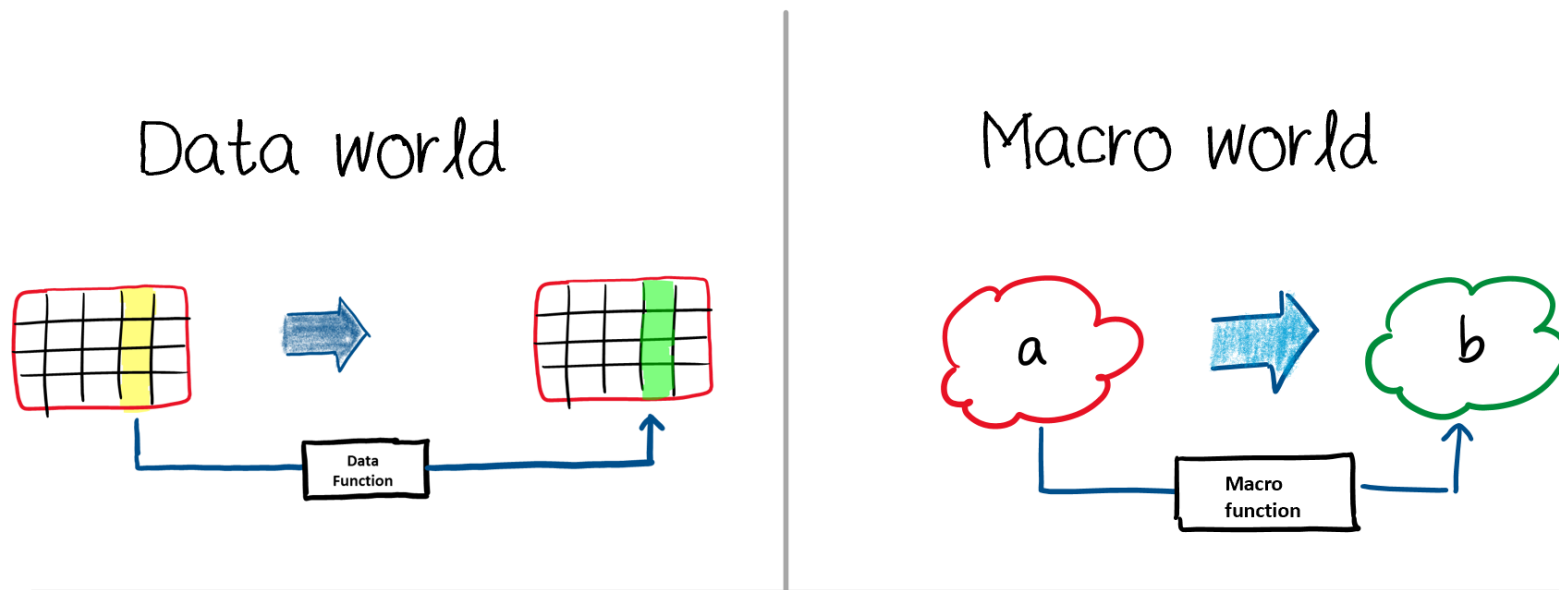
-
- There are three rules of ampersand resolution:
 - resolution happens from left to right,
 - all consecutive double ampersands are resolved into a single ampersand, and
 - the resolution continues till all ampersands are resolved.
 - Consecutive ampersands are used to obtain the innermost value of a nested macro variable.

SAS MACRO



- SAS macros are useful in identifying repetitive code with patterns that can be replicated.
- They can be used to make SAS programming more efficient and productive.
- Macros are abstracted blocks of code that can be replaced and reused.
- A specific macro name can be used to identify a block of code that is repetitive in nature.
- By magnetizing code, you can convert it into a single module of code that can be used to replicate all repetitive code by replacing only the changing parts.
- The changing parts are replaced by macro variables that are defined within a macro using the %macro keyword followed by the name of the macro, in parentheses the macro variables to be used in the section of code identified as the repetitive code.

MACRO FUNCTIONS



Functions on char-like values

- Uppcase
- Lowcase
- Substr
- Length

Functions on num-like values

- Eval
- sysevalf

- Macro functions operate on macro variables, which exist in the exclusive macro world
- Macros extensively interact with the data world and offer benefits in using them
- Macro functions have a slightly different syntax than functions for data sets
- Macro functions are denoted by a percent sign before the function name, eg: %uppercase



made2sticklearning.com

Course created and instructed by:

Aslam Khan



<https://www.linkedin.com/in/aslamkhan-gemp-gmp/>

@AslamKhan54321



Order your course completion certificate from

sas.made2sticklearning.com

Order your course completion certificate after you have completed this training (all lectures, quizzes, assignments and practice exams)
Go to sas.made2sticklearning.com (scroll to the bottom of the page for order button)