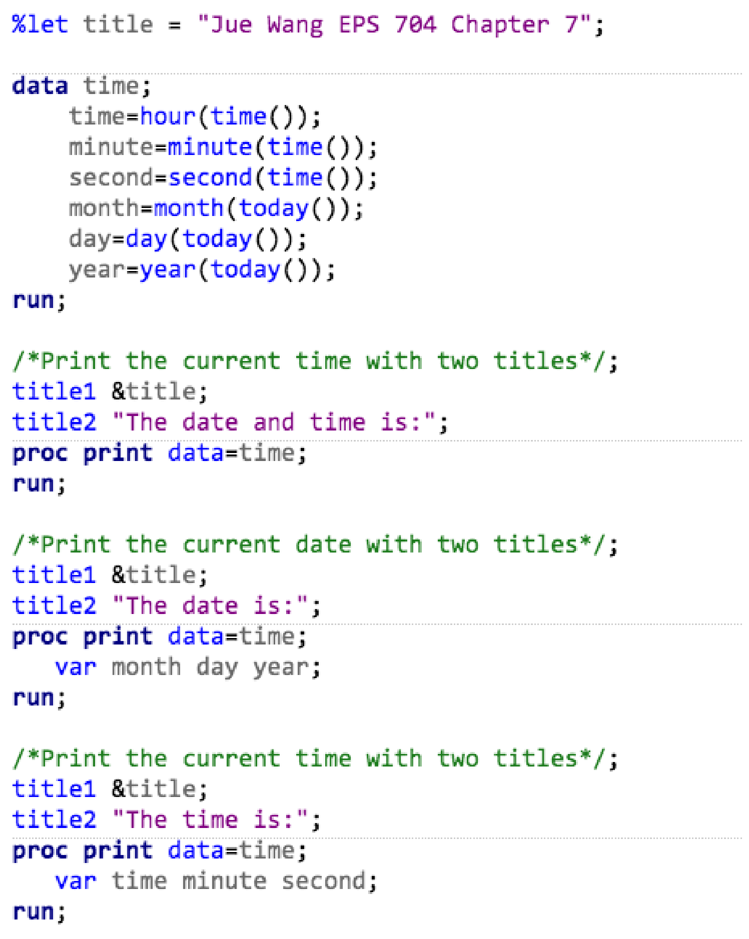
Chapter 7 Macro in SAS

Macros are particularly useful when we want to make SAS programs more flexible -- allow them to be used in different situations without having to rewrite the entire program. Macro variables also provide a useful method for passing information from one DATA step to another.

Let’s look at an example first – create a title and re-use it throughout the program.



```
%let title = "Jue Wang EPS 704 Chapter 7";

data time;
    time=hour(time());
    minute=minute(time());
    second=second(time());
    month=month(today());
    day=day(today());
    year=year(today());
run;

/*Print the current time with two titles*/;
title1 &title;
title2 "The date and time is:";
proc print data=time;
run;

/*Print the current date with two titles*/;
title1 &title;
title2 "The date is:";
proc print data=time;
   var month day year;
run;

/*Print the current time with two titles*/;
title1 &title;
title2 "The time is:";
proc print data=time;
   var time minute second;
run;
```

Create an object called **title**, and its value is: **Jue Wang EPS 704 Chapter 7**.

Create a temporary dataset with system time and date as variables.

After created the **title** object, we can call it instead of inputting the entire phrase.

Note: We can actually just define TITLE1 once.

General functions in SAS

a. Title statement: The title line with the highest number appears on the bottom line. If you omit a number, SAS assumes a value of 1. Therefore, you can specify TITLE or TITLE1 for the first title line. The index ranges from 1 to 10.

- A blank title (e.g., title3 ;) will generate a blank line after title2.
- Can make it fancier by modifying font, size, color, etc.
- Look up the examples in the SAS Help Center: https://documentation.sas.com/?docsetId=lestmtsglobal&docsetTarget=p10gcmrmf83iaxn1ilrx4pra969n.htm&docsetVersion=9.4&locale=en

b. Time and date

- The TIME function finds the system time which is written as one number.
- The HOUR, MINUTE, and SECOND functions pulls their corresponding values out of the TIME() value.
- The TODAY function finds the number of days since January 1, 1960.
- The MONTH, DAY, and YEAR functions pull their corresponding values out of the TODAY() value.

c. Mathematical functions

Commonly used math functions

Function Name	**Definition**	**Example (SAS commands)**
Abs	Absolute value	y = abs(x)
Int	Integer (return the integer part)	y = int(x)
Log	Natural log	y = log(x)
Log10	Log base 10	y = log10(x)
Round	Rounds the argument to a specified level	y = round(x, .01)
Sqrt	Square root	y = sqrt(x)

7.1 What is Macro (or Macro facility) in SAS? How to define Macro variables?

A Macro is a chunk of programming codes, which consists of system variables, statements, and functions that are directly processed by SAS via the macro processor. This facility allows us to package large (or small) amounts of text into units or modules that contain macro names. Once it's all set up, we simply need to work with the macro names instead of modifying the text.
Benefits of using SAS macro

- With macros, you can make one small change in your program and have SAS echo that change throughout your program.
- Macros allow you to write a piece of code once and use it over and over, in the same program or in different programs.
- You can make your programs data driven, letting SAS decide what to do based on actual data values.

Two key delimiters trigger macro processor activity

- **%name**: refers to a macro
- **&name**: refers to a macro variable

We will learn the following pre-defined macros by SAS

- **%LET**: Statements used to define a constant throughout the program.
- **%PUT**: Write to the SAS LOG window.
- **%INCLUDE**: Include another SAS program's code in the current one.

Note: SAS macro variables can be defined and used anywhere in a SAS program, except in **DATALINES**.

7.2 The most basic macro -- %LET

This allows you to declare a constant to be used throughout the program. The syntax is ***%LET macro-variable-name = value;***

- The ***macro-variable-name*** is the name of the macro variable which is subject to standard SAS naming convention.
- The ***value*** can be any string or macro expression. It is stored as a character – quotation marks are NOT needed. Any characters between the equal sign (=) and the semicolon (;) are considered as the ***value*** of the macro variable. The ***value*** remains constant/fixed throughout the program.
- We usually put all **%LET** statements at the beginning of the program for ease of modifying.

Example 2: Define the sample size with **%LET.**

Suppose we want to randomly generate 10 values from a standard normal distribution. Later, we want to generate 20, 30, and 50 values from a standard normal distribution. The only thing changes here is the sample size. The data generation procedure remains the same.

Therefore, we can use the macro **%LET** to create a macro variable for sample size. Later, we only need to change the value of the macro variable in the **%LET** statement.

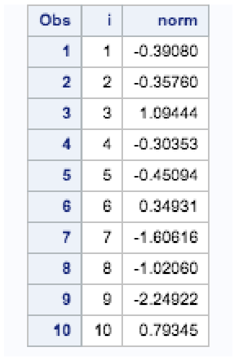
`%let n=10000;`	**%let** is a macro. **n** is the macro variable. The **value** of this macro variable is 10.

`data normal;`
`    do i=1 to &n;`
`        norm=normal(0);`
`        output;`
`    end;`
	Data generation. The normal(0) will return a random number from standard normal distribution.
	Do… To… END does replications for the same procedure. &n defines the total number of iterations.

`proc print data=normal;`
`run;`
	Then we print out the dataset.

Note: The OUTPUT statement is important – it stores the generated values to dataset normal. Without it, the previously generated value will be replaced by the current one.

Results:



Obs	i	norm
1	1	-0.39080
2	2	-0.35760
3	3	1.09444
4	4	-0.30353
5	5	-0.45094
6	6	0.34931
7	7	-1.60616
8	8	-1.02060
9	9	-2.24922
10	10	0.79345

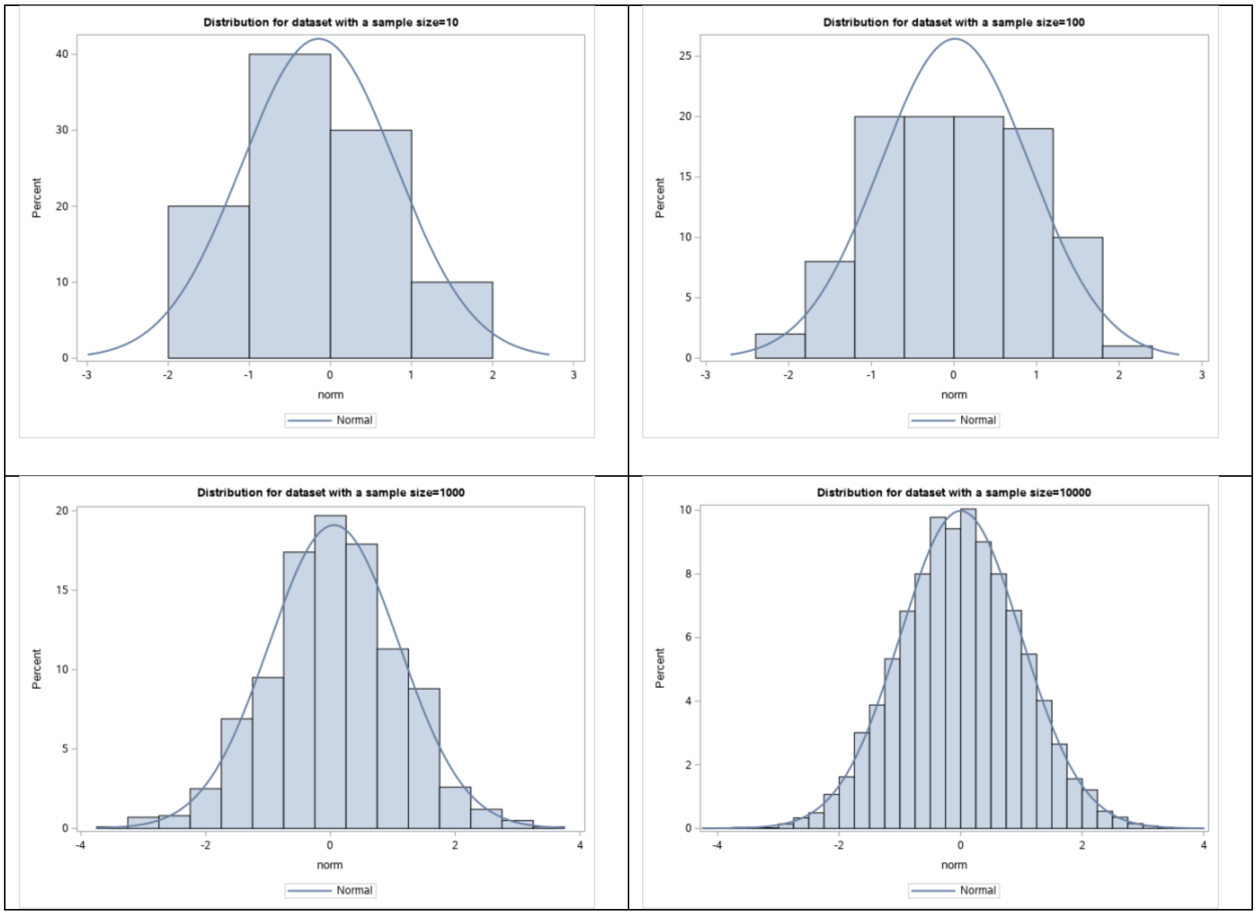
Exercise 1. Let's continue this example by changing the sample size to be 100, 1,000, and 10,000. Then we create histogram for each dataset and compare.

- The following chunk of codes will be re-run after changing the value of macro variable in the **%LET** statement each time.
- In this exercise, you will see we use the macro variable (&n) for at multiple places.
- The function *normal(0)* generates observations/numbers from a standard normal distribution, i.e., N(0, 1).

```
/*We create different datasets by using macro variable in creating dataset name*/;
data normal&n;
    do i=1 to &n;
        norm=normal(0);
        output;
    end;

/*Produce histograms to compare the distributions of generated datasets*/;
title2 "Distribution for dataset with a sample size=&n";
proc sgplot data=normal&n;
  histogram norm;
  density norm;
run;
```

Results:



7.3 Macro %PUT

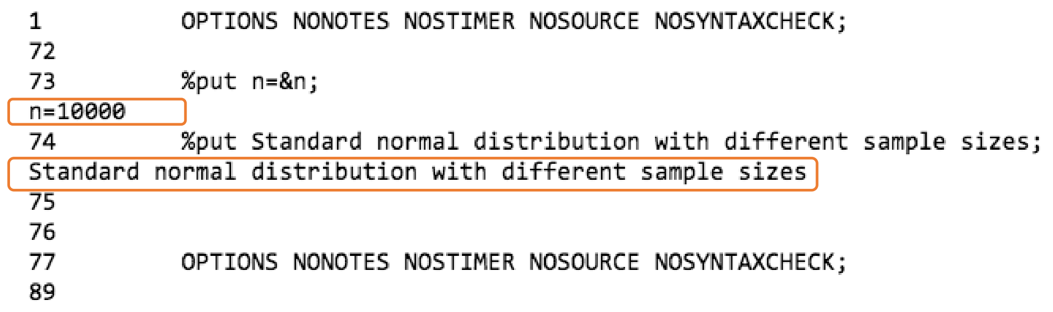
This macro writes the value to the SAS LOG window. It can be helpful for debugging a program.

Example 3

```
/*Example 3 Write to LOG window using %PUT*/;
%put n=&n;
%put Standard normal distribution with different sample sizes;
```

Note: You can write anything to the LOG window using **%PUT**. Quotation marks are not needed.

LOG window:



```
 1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 72
 73         %put n=&n;
 n=10000
 74         %put Standard normal distribution with different sample sizes;
 Standard normal distribution with different sample sizes
 75
 76
 77         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 89
```

7.4 Macro %INCLUDE

This macro allows you to include another program into the one you are currently using. Suppose a SAS program, named *small_prog.sas*, contains only the following commands.

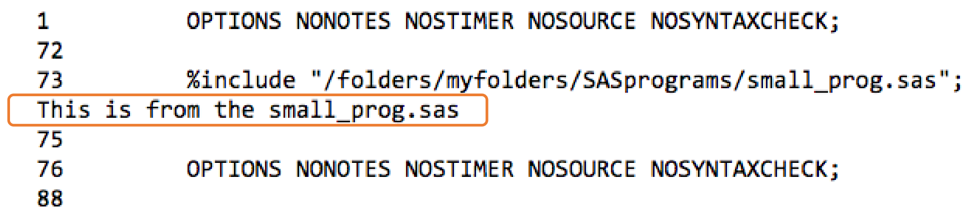
%put This is from the small_prog.sas;

In the active SAS program, we use **%INCLUDE** to call and execute the included program.

```
/*Example 4 Include another SAS program using %INCLUDE*/;
%include "/folders/myfolders/SASprograms/small_prog.sas";
```

Therefore, it will run the %PUT macro and writes to the LOG window.

LOG window:



```
 1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 72
 73         %include "/folders/myfolders/SASprograms/small_prog.sas";
 This is from the small_prog.sas
 75
 76         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 88
```

Note: When using %INCLUDE in SAS University Edition, we need to begin the file path with “/folders”.

7.5 Write your own Macro

The template for a user-defined Macro.

```
%macro  macro-name

<SAS Statements>

%mend macro-name

*Begin main program
…
```

Then we can invoke the SAS macro anywhere in the main program. (Similar to R functions)

Example 5: Write a Macro to print the first 10 observations for any dataset.

Step 1: Define the Macro PRINTIT

```
/*Example 5 Write a simple Macro*/;
%macro printit(dataset);
title "First 10 observations of &dataset";
proc print data=&dataset (obs=10);
run;
%mend printit;
```

- The name of macro is *printit*.
- *dataset* is a macro variable which only works inside of this macro.
- Within the macro, we use & to identify the macro variable.
    - Customize the title to show the name of specific dataset.
    - Identify the dataset to be printed in PROC PRINT line.

Step 2: Execute the Macro PRINTIT

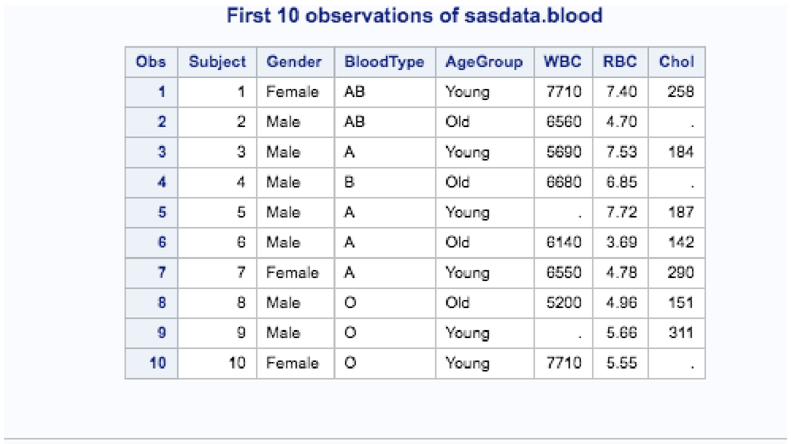
- A Macro call should be outside of a DATA step or any PROC
- This call is in the main program.

```
%printit(sasdata.blood);
%printit(sashelp.stocks);
```

- Print the first 10 observations of dataset blood
- Print the first 10 observations of dataset stocks

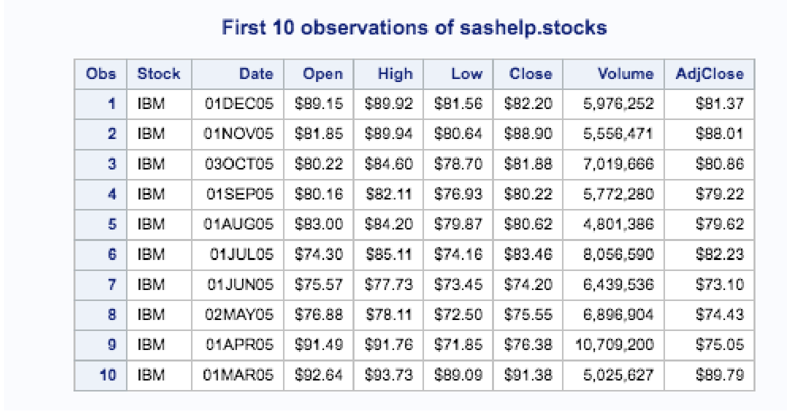
Note: Basically, SAS replaces the macro variable with the specified value in the SAS statements.

Results:



First 10 observations of sasdata.blood

Obs	Subject	Gender	BloodType	AgeGroup	WBC	RBC	Chol
1	1	Female	AB	Young	7710	7.40	258
2	2	Male	AB	Old	6560	4.70	.
3	3	Male	A	Young	5690	7.53	184
4	4	Male	B	Old	6680	6.85	.
5	5	Male	A	Young	.	7.72	187
6	6	Male	A	Old	6140	3.69	142
7	7	Female	A	Young	6550	4.78	290
8	8	Male	O	Old	5200	4.96	151
9	9	Male	O	Young	.	5.66	311
10	10	Female	O	Young	7710	5.55	.



First 10 observations of sashelp.stocks

Obs	Stock	Date	Open	High	Low	Close	Volume	AdjClose
1	IBM	01DEC05	$89.15	$89.92	$81.56	$82.20	5,976,252	$81.37
2	IBM	01NOV05	$81.85	$89.94	$80.64	$88.90	5,556,471	$88.01
3	IBM	03OCT05	$80.22	$84.60	$78.70	$81.88	7,019,666	$80.86
4	IBM	01SEP05	$80.16	$82.11	$76.93	$80.22	5,772,280	$79.22
5	IBM	01AUG05	$83.00	$84.20	$79.87	$80.62	4,801,386	$79.62
6	IBM	01JUL05	$74.30	$85.11	$74.16	$83.46	8,056,590	$82.23
7	IBM	01JUN05	$75.57	$77.73	$73.45	$74.20	6,439,536	$73.10
8	IBM	02MAY05	$76.88	$78.11	$72.50	$75.55	6,896,904	$74.43
9	IBM	01APR05	$91.49	$91.76	$71.85	$76.38	10,709,200	$75.05
10	IBM	01MAR05	$92.64	$93.73	$89.09	$91.38	5,025,627	$89.79

7.6 Simulating Data in SAS

Example 6: Simulate data from the Standard normal distribution

a. Simulate data using a DATA step with Do loops.

```
data normal (keep=x);
    call streaminit(1234);

    do i=1 to 100;
        x=rand("Normal");
        output;
    end;
run;
```

- KEEP: the variables that will be saved to this dataset.
- STREAMINIT function: set a seed value – can be any integer. Seeds are used for random number generation.
- Do … to … end: iterate this procedure 100 times
- RAND function: generate a random value from a distribution (the standard normal distribution). rand ("Normal") is the same as normal(0)

A few notes.

- The seed number realizes the random generation. It is used to initialize a pseudorandom number generator.

- If you change the seed value, you will get a different random sample. In simulation studies, we usually use a fixed seed number so that the results can be reproduced. We also need to report the seed number when publishing the study so that others can replicate our analyses.
- In the SAS Help Center, you can find a list of distributions that work with the RAND function: https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p0fpeei0opypg8n1b06qe4r040lv.htm&docsetVersion=9.4&locale=en

b. Simulate data using PROC IML.

IML stands for Interactive Matrix Language. We can generate random samples by using RANDGEN function in SAS/IML software.

Example 7: RANDGEN in PROC IML

Suppose we want to generate 100 random numbers from a standard normal distribution.

```
%let N = 100;

proc iml;
call randseed(1234);
      /*Specify a seed number that initializes the random number stream; same use as STREAMINIT in a DATA step.*/;
x = j(&N, 1);
      /*The J function defines a matrix J(r, c) with N number of rows and 1 column --> vector; assign it to x.*/;
call randgen (x, "Normal");
     /*RANDGEN: produces random numbers from a specified distribution to fill in the matrix x.*/;
print x;
     /*Print the matrix x to the RESULTS window.*/;
run;
```

If we want to save the generated values to a SAS dataset, we use CREATE, APPEND, and CLOSE statements.

```
proc iml;
call randseed(1234);
x = j(&N, 1);
call randgen (x, "Normal", 0, 1);
create normal;    /*Define a dataset named normal.*/;
append var{x};    /*Add x as a variable to the dataset normal.*/;
close normal;     /*Tell SAS that we are done with writing this dataset.*/;
run;

proc print data=normal;
run;
```

Example 8: Generate 20 random numbers with UNIFORM(REPEAT()) in PROC IML.

```
proc iml;
y = repeat(0,20,1);
     /*The REPEAT function creates a matrix of repeated values -> A 20 by 1 ZERO matrix (all entries are 0)*/;
u=uniform(repeat(0,20,1));
     /*uniform(0) produces a random number between 0 and 1 --> it generates 20 random numbers*/;
do i = 1 to 20;
    if u[i]>0.5 then y[i]=1;  /*If u[i] > 0.5, assign 1 to y[i]; otherwise, leave it as 0.*/;
end;
print u y;
run;
```