Advanced SAS Interview Questions and Answers

1. Two ways to select every second row in a data set

```
data example;
set sashelp.class;
if mod(_n_,2) eq 0;
run;
```

MOD Function returns the remainder from the division of the first argument by the second argument. _N_ corresponds to each row. The second row would be calculated like (2/2) which returns **zero** remainder.

```
data example1;
do i = 2 to nobs by 2;
set sashelp.class point=i nobs=nobs;
output;
end;
stop;
run;
```

2. How to select every second row of a group

Suppose we have a table **sashelp.class**. We want every second row by variable 'sex'.

```
proc sort data = sashelp.class;
by sex;
run;
data example2 (drop = N);
set sashelp.class;
by sex;
if first.sex then N = 1;
else N +1;
if N = 2 then output;
```

run;

3. How to calculate cumulative sum by group

Create Sample Data

```
data abcd;
input x y;
cards;
1 25
1 28
1 27
2 23
2 35
2 34
3 25
3 29
;
run;
```

x	y	z	z1
1	25	1	25
1	28	2	53
1	27	3	80
2	23	1	23
2	35	2	58
2	34	3	92
3	25	1	25
3	29	2	54

Cumulative Sum by Group

Cumulative Sum by X

```
data example3;
set abcd;
if first.x then z1 = y;
else z1 + y;
by x;
run;
```

4. Can both WHERE and IF statements be used for subsetting on a newly derived variable?

```
data example4;
set abcd;
z = x*y;
if z <=50;
run;
```

```
data example5;
set abcd;
z = x*y;
where  z <=50;
run;
```

SAS : WHERE vs. IF

No. Only **IF statement** can be used for subsetting when it is based on a newly derived variable. **WHERE** statement would return an error "newly derived variable is not on file".

Please note that **WHERE** Option can be used for subsetting on a newly created variable.

```
data example4 (where =(z <=50));
set abcd;
z = x*y;
run;
```

5. Select the Second Highest Score with PROC SQL

```
data example5;
input Name $ Score;
cards;
sam 75
dave 84
sachin 92
ram 91
;
run;
proc sql;
select *
from example5
where score in (select max(score) from example5 where score not in (select max(score)
from example5));
quit;
```

6. Two ways to create a macro variable that counts the number of observations in a dataset

```
data _NULL_;
if 0 then set sashelp.class nobs=n;
call symputx('totalrows',n);
stop;
run;
%put nobs=&totalrows.;
proc sql;
select count(*) into: nrows from sashelp.class;
quit;
%put nobs=%left(&nrows.);
```

7. **Suppose you have data for employees. It comprises of employees' name, ID and manager ID. You need to find out manager name against each employee ID.**

Input			Output
Name	ID	ManagerID	Manager
Robert	456	.	
Cook	383	222	William
Daniel	777	222	William
Smith	123	456	Robert
William	222	456	Robert

SQL: Self Join

Create Sample Data

```
data example2;
input Name $ ID ManagerID;
cards;
Smith 123 456
Robert 456 .
William 222 456
Daniel 777 222
Cook 383 222
;
run;
```

SQL Self Join

```
proc sql;
create table want as
select a.*, b.Name as Manager
```

from example2 as a left join example2 as b

on a.managerid = b.id;

quit;

Data Step : Self Join

proc sort data=example2 out=x;

by ManagerID;

run;

proc sort data=example2 out=y (rename=(Name=Manager ID=ManagerID

ManagerID=ID));

by ID;

run;

data want;

merge x (in= a) y (in=b);

by managerid;

if a;

run;

8. Create a macro variable and store TomDick&Harry

Issue : When the value is assigned to the macro variable, the ampersand placed after**TomDick** may cause SAS to interpret it as a macro trigger and an warning message would be occurred.

%let x = %NRSTR(TomDick&Harry);

%PUT &x.;

%NRSTR** function is a macro quoting function which is used to hide the normal meaning of special tokens and other comparison and logical operators so that they appear as constant text as well as to mask the **macro triggers (%, &).

9. Difference between %STR and %NRSTR

Both %STR and %NRSTR functions are macro quoting functions which are used to hide the normal meaning of special tokens and other comparison and logical operators so that they appear as constant text. ***The only difference is %NRSTR can mask the macro triggers (%, &) whereas %STR cannot.***

10. How to pass unmatched single or double quotations text in a macro variable

%let eg = %str(%'x);

%let eg2 = %str(x%");

%put ⪚

%put &eg2;

If the argument to %STR or %NRSTR contains an single or double quotation mark or an unmatched open or close parenthesis, precede each of these characters with a **% sign**.

11. How can we use COUNTW function in a macro

%let cntvar = %sysfunc(countw(&nvar));

There are several useful Base SAS function that are not directly available in Macro,**%Sysfunc** enables those function to make them work in a macro.

%let x=temp;

%let n=3;

%let x3=result;

%let temp3 = result2;

What %put &&x&n; and %put &&&x&n; would return?

1. &&x&n : Two ampersands (&&) resolves to one ampersand (&) and scanner continues and then N resolves to 3 and then &x3 resolves to **result**.

2. &&&x&n : First two ampersands (&&) resolves to & and then X resolves to temp and then N resolves to 3. In last, &temp3 resolves to **result2**.

12. How to reference a macro variable in selection criteria

*Use **double quotes** to reference a macro variable in a selection criteria.**Single quotes** would not work.*

Wrong Code (×)

```
%macro simple (criteria=);
data temp;
set sashelp.heart;
where sex = '&criteria';
run;
%mend;

%simple(criteria = Female);
```

Right Code

```
%macro simple (criteria=);
data temp;
set sashelp.heart;
where sex = "&criteria";
run;
%mend;

%simple(criteria = Female);
```

SAS : Reference Macro Variable

13. How to debug %IF %THEN statements in a macro code

***MLOGIC** option will display how the macro variable resolved each time in the LOG file as TRUE or FALSE for %IF %THEN.*

14. Difference between %EVAL and %SYSEVALF functions

Both %EVAL and %SYSEVALF are used to perform mathematical and logical operation with macro variables. **%let last = %eval (4.5+3.2);** returns error as %EVAL cannot perform arithmetic calculations with operands that have the floating point values. It is when the **%SYSEVALF** function comes into picture.

```
%let last2 = %sysevalf(4.5+3.2);
%put &last2;
```

15. What would be the value of i after the code below completes

```
data test;
set temp;
array nvars {3} x1-x3;
do i = 1 to 3;
if nvars{i} > 3 then nvars{i} =.;
end;
run;
```

Answer is 4. It is because when the first time the loop processes, the value of count is 1; the second time, 2; and the third time, 3. At the beginning of the fourth iteration, the value of count is 4, which is found to be greater than the stop value of 3 so the loop stops. However, the value of i is now 4 and not 3, the last value before it would be greater than 3 as the stop value.

16. How to compare two tables with PROC SQL

The **EXCEPT** operator returns rows from the first query that are not part of the second query.

```
proc sql;
select * from newfile
except
select * from oldfile;
quit;
```

17. Selecting Random Samples with PROC SQL

The **RANUNI** and **OUTOBS** functions can be used for selecting **N** random samples. The**RANUNI** function is used to generate random numbers.

```
proc sql outobs = 10;
create table tt as
select * from sashelp.class
order by ranuni(1234); quit;
```

18. How to use NODUPKEY with PROC SQL

In PROC SORT, NODUPKEY option is used to remove duplicates based on a variable. In SQL, we can do it like this :

```
proc sql noprint;
create table tt (drop = row_num) as
select *, monotonic() as row_num
from readin
group by name
having row_num = min(row_num)
order by ID;
quit;
```

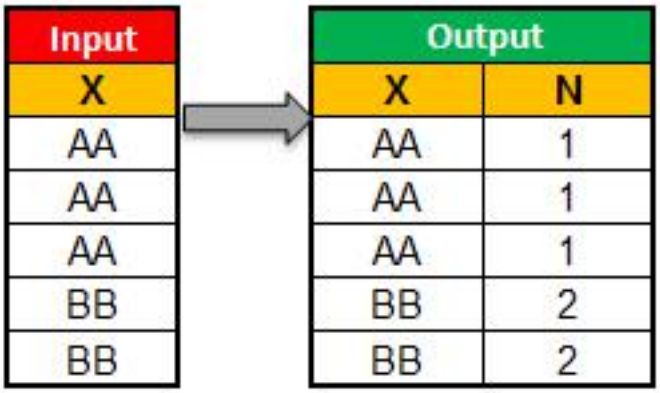
19. How to make SAS stop macro processing on Error

20. Count Number of Variables assigned in a macro variables

```
%macro nvars (ivars);
%let n=%sysfunc(countw(&ivars));
%put &n;
%mend;
%nvars (X1 X2 X3 X4);
```

21. Two ways to assign incremental value by group

See the snapshot below -



Input	Output	
X	X	N
AA	AA	1
AA	AA	1
AA	AA	1
BB	BB	2
BB	BB	2

Advanced SAS Interview Questions

Prepare Input Data

```
data xyz;
input x $;
cards;
AA
AA
AA
BB
BB
;
run;
```

Data Step Code

```
data example22;
set xyz;
if first.x then N+1;
by x;
proc print;
run;
```

PROC SQL Code

```
proc sql;
select a.x, b.N from xyz a
inner join
(select x, monotonic() as N
from (
select distinct x
from xyz)) b
on a.x=b.x;
quit;
```

22. Prepare a Dynamic Macro with %DO loop

23. Write a SAS Macro to extract Variable Names from a Dataset

```
*Selecting all the variables;
proc sql noprint;
select name into : vars separated by " "
from dictionary.columns
where LIBNAME = upcase("work")
and MEMNAME = upcase("predata");
quit;
```

*The **DICTIONARY.COLUMNS** contains information such as name, type, length, and format, about all columns in the table.* **LIBNAME :** Library Name, **MEMNAME :** Dataset Name

```
%put variables = &vars.;
```

24. How would DATA STEP MERGE and PROC SQL JOIN works on the following datasets shown in the image below?

Sample I

ID	Info
1	3123
1	1234
2	7482
2	8912
3	1284

Sample II

ID	Info2
1	4444
1	5555
1	8989
2	9099
2	8888
3	8989

Many to Many Merge

The DATA step does not handle many-to-many matching very well. When we perform many to many merges. the result should be a cartesian (cross) product. For example, if there are three records that match from one contributing data set to two records from the other, the resulting data set should have 3 × 2 = 6 records. Whereas, PROC SQL creates a cartesian product in case of many to many relationship.

25. Two ways to create a blank table

Copy structure of existing table

```
PROC SQL;
CREATE TABLE EXAMPLE2 LIKE TEMP;
QUIT;
```

Enforce FALSE condition in Selection Criteria

```
PROC SQL NOPRINT;
CREATE TABLE EXAMPLE2 AS
SELECT * FROM TEMP
WHERE 1=0;
QUIT;
```

26. How to insert rows in a table with PROC SQL

27. Difference between %LOCAL and %GLOBAL

%LOCAL *is used to create a local macro variable during macro execution. It gets removed when macro finishes its processing.*

%GLOBAL *is used to create a global macro variable and would remain accessible till the end of a session . It gets removed when session ends.*

28. Write a macro with CALL EXECUTE

29. Write a macro to split data into N number of datasets

Suppose you are asked to write a macro to split large data into 2 parts (not static 2). In the macro, user should have flexibility to change the number of datasets to be created.

```
%macro split(inputdata=, noofsplits=2);
data %do i = 1 %to &noofsplits.; split&i.
%end;;
retain x;
set &inputdata. nobs=nobs;
if _n_ eq 1 then do;
if mod(nobs,&noofsplits.) eq 0
then x=int(nobs/&noofsplits.);
else x=int(nobs/&noofsplits.)+1;
end;
if _n_ le x then output split1;
%do i = 2 %to &noofsplits.;
else if _n_ le (&i.*x)
then output split&i.;
```

```
%end;
run;
%mend split;
%split(inputdata=temp, noofsplits=2);
```

30. Store value in each row of a variable into macro variables

```
data _null_;
set sashelp.class ;
call symput(cats('x',_n_),Name);
run;
%put &x1. &x2. &x3.;
```

The CATS function is used to concatenate 'x' with _N_ (row index number) and removes leading and trailing spaces to the result.