TOP 10 (OR MORE) WAYS TO OPTIMIZE YOUR SAS CODE

Handy Tips for the Savvy Programmer

SAS PROGRAMMING BEST PRACTICES

- Create Readable Code
- Basic Coding Recommendations
  - Efficiently choosing data for processing
  - When to use indexes
  - Other general recommendations
- Developing Code

CREATE READABLE CODE

Tips for creating code that you and your co-workers will find easy to read and understand.

1. COMMENT, COMMENT, COMMENT!

Method 1:

```
/* create summary report*/
proc means data=new;
   more statements here;
run;
```

Method 2:

```
*create summary report;
proc means data=old;
   more statements here;
run;
```

Note: Method 1 may also be helpful when developing and debugging code.

1. COMMENT, COMMENT, COMMENT!

Method 1:

```
/*
data new;
   set old;
run;
*/
proc means data=new;
   more statements here;
run;
```

Efficiency consideration: every submission of the DATA step re-creates the SAS data.

2. USE RECOMMENDED FORMATTING FOR SAS CODE

Do this:

```
data new;
  set old;
Run;
proc means data=new;
  var newvar;
  class year;
run;
```

Not this:

```
data new; set old; run;
proc means data=new;
var newvar; class year;
run;
```

3. USE DESCRIPTIVE NAMES

Do this:

```
data salaryinfo2012;
  set salaryinfo2011;
  newsalary=
      oldsalary+increase;
run;
```

Not this:

```
data new;
  set old;
  z=x+y;
run;
```

Note: If you are forced to use a project’s naming conventions, then use block comments with variable name descriptions to help describe the variables.

4. USE UNDERSCORES OR CAMEL CASE TO CREATE DESCRIPTIVE NAMES

Camel Case

```
data salaryInfo2012;
  set salaryInfo2011;
  newSalary=
      oldSalary+increase;
run;
```

Underscores

```
data salary_info2012;
  set salary_info2011;
  new_salary=
       old_salary+increase;
run;
```

SAS names:

- Can be 32 characters long.
- Must start with a letter or underscore, continuing with numbers, letters or underscores.
- Can be uppercase, lowercase or mixed case.
- Are not case sensitive.

5. PUT ALL “GLOBAL” STATEMENTS AT THE BEGINNING OF YOUR CODE

Libname statements, system options, and title statements are easier to find (and change, if necessary) if they are all in one place.

BASIC CODING RECOMMENDATIONS

Basic coding recommendations to increase the efficiency of your SAS programs.

6. MINIMIZE THE NUMBER OF TIMES YOU READ YOUR DATA

Do this:

```
data a b c;
  set old;
  if condition then
     output a;
  else if condition then
     output b;
  else if condition then
     output c;
run;
```

Not this:

```
data a;
    set old;
    [more code]
run;
data b;
    set old;
    [more code]
run;
data c;
    set old;
    [more code]
run;
```

7. LIMIT THE NUMBER OF TIMES YOU SORT YOUR DATA

If you think the incoming data is already sorted, use the **presorted** option on your SORT statement; the sort order will be verified.

```
data new;
     infile ‘rawdata.dat’;
     input ID $ 1-4 name $ 5-25 salary 26-35;
run;
```

```
proc sort data=new out=new_sorted presorted;
    by ID;
run;
```

7A. LIMIT THE NUMBER OF TIMES YOU SORT YOUR DATA

When creating an SQL view, avoid including an ORDER BY clause in the view, as the data will need to be sorted every time the view is used.

```
proc sql;
   create view sql.new as
      select *
        from sql.old
        order by firstvar;

proc print data=sql.new;
run;
```

The PROC PRINT or any other procedure/DATA step that uses the view will execute the stored SQL query, including the ORDER BY.

7A. LIMIT THE NUMBER OF TIMES YOU SORT YOUR DATA

In our example, the SQL view stores the following query:

```
proc sql;
   create view sql.new as
      select *
        from sql.old
        order by firstvar;
```

Stored in SQL view

```
proc print data=sql.new;
run;
```

View is executed

8. USE IF-THEN-ELSE INSTEAD OF IF-IF-IF

Do this:

```
data new;
  set old;
  if condition then
     some action;
  else if condition then
     some other action;
  else if condition then
     some other action;
run;
```

Not this:

```
data new;
  set old;
  if condition then
     some action;
  if condition then
     some other action;
  if condition then
     some other action;
run;
```

Note: It is recommended that you use a SELECT group rather than a series of IF-THEN statements when you have a long series of mutually exclusive conditions.

8A. USE IF-THEN-ELSE INSTEAD OF IF-IF-IF

IF THEN

- When there are few conditions to check.
- The values are not uniformly distributed.
- The values are character or the values are discrete numeric data.

SELECT

- When there is a long series of mutually exclusive conditions.
- The values are numeric and are uniformly distributed.

9. ORDER IF THEN CONDITIONS IN DESCENDING ORDER OF PROBABILITY

```
data new;
  set old;
  if condition occurring most often then
     some action;
  else if condition then
     some other action;
  else if condition then
     some other action;
run;
```

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

Do This:

```
data new;
   set old (drop=category
      type value ...);
   more statements here;
run;
```

Not This:

```
data new;
   set old;
   more statements here;
run;
```

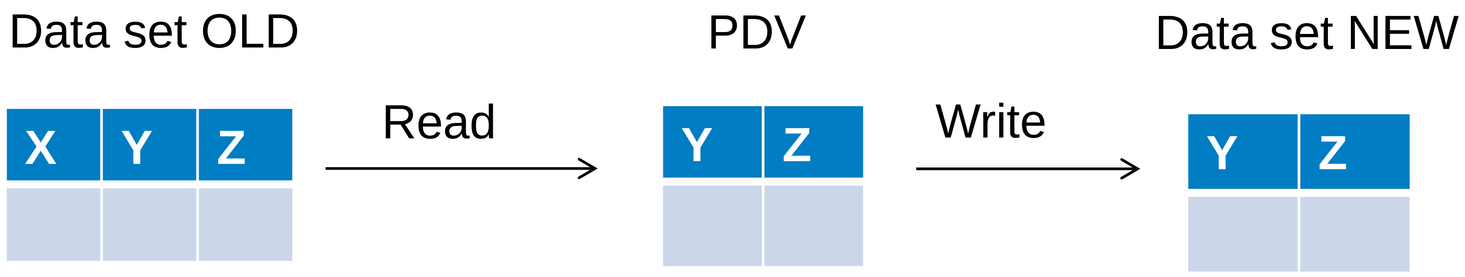
Variations:

- Use the keep= option if you need to keep more variables than you need to drop!
- Use both keep= and drop= options to control variables on both the incoming and outgoing sides!
- Keep= and drop= options can be used in PROC steps, too!

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

When you use DROP=/KEEP= on the SET statement, you affect what is **read from** the existing SAS data set.

```
data new;
  set old(drop=x);
run;
```

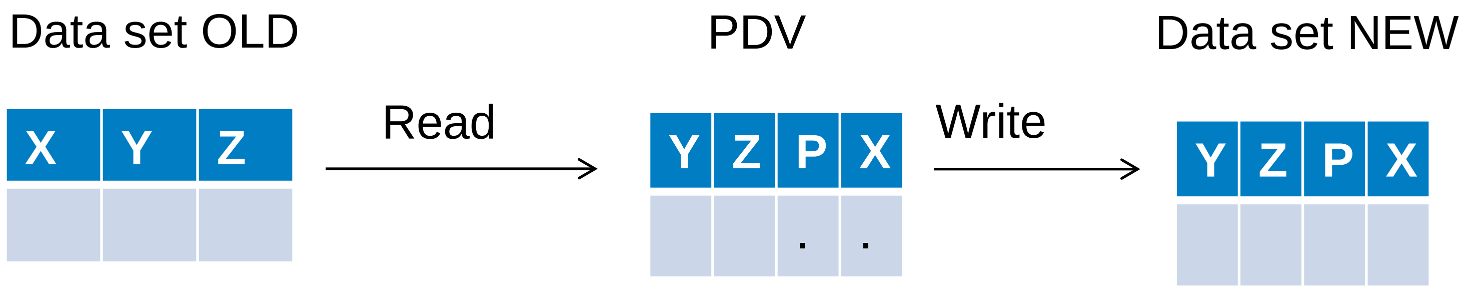


Note: Variables not read into the PDV are not available for processing. Consider the following assignment statement:
p=x+y;

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

The variables P and X are added to the PDV at compile time when SAS encounters the assignment statement.

```
data new;
  set old(drop=x);
  p=x+y;
run;
```

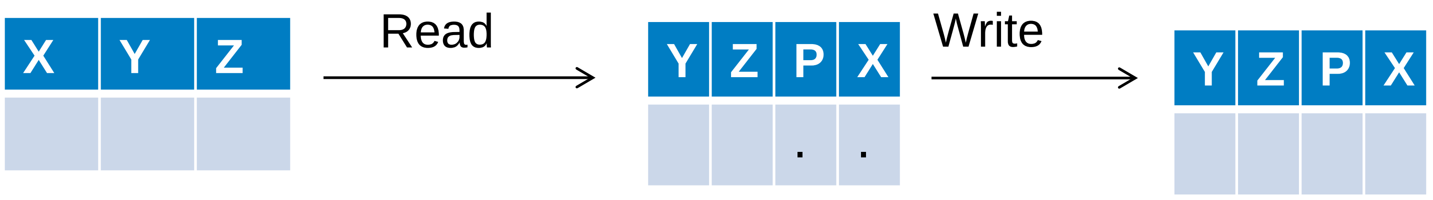


Since the variables X and P are being created due to the assignment statement, they both receive initial values of missing.

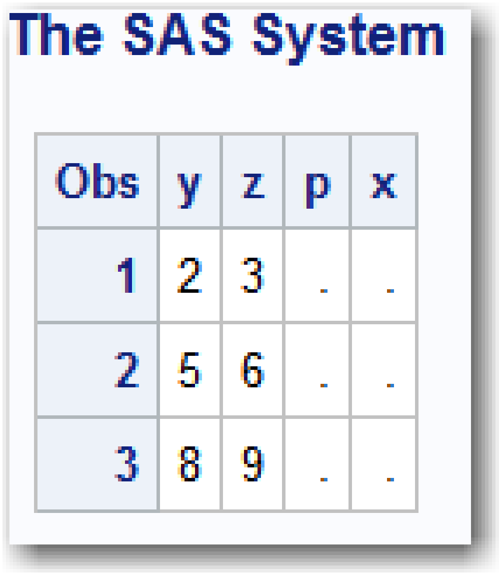
10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

The variables P and X are added to the PDV at compile time when SAS encounters the assignment statement. X is considered to be a ‘new’ variable.

```
data new;
  set old(drop=x);
  p=x+y;           ------------>   p=.+y;
run;
```



10. OUTPUT


The SAS System

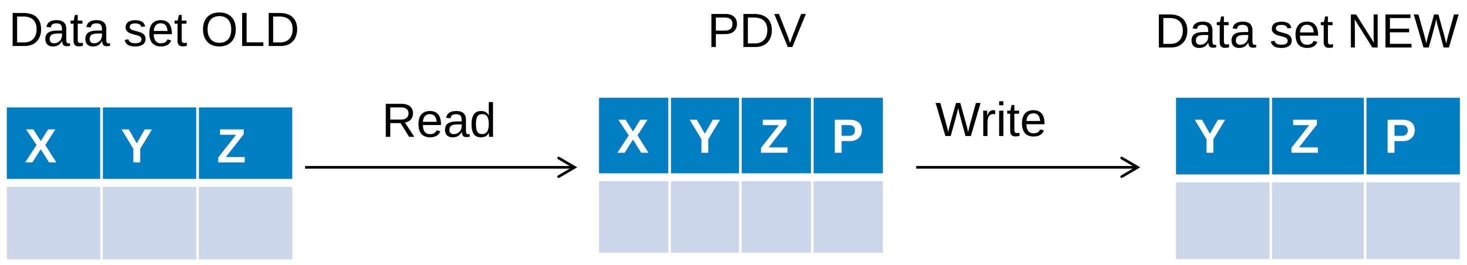
Obs	y	z	p	x
1	2	3	.	.
2	5	6	.	.
3	8	9	.	.

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

When you use DROP=/KEEP= on the DATA statement, you affect what is **written to** the output SAS data set.

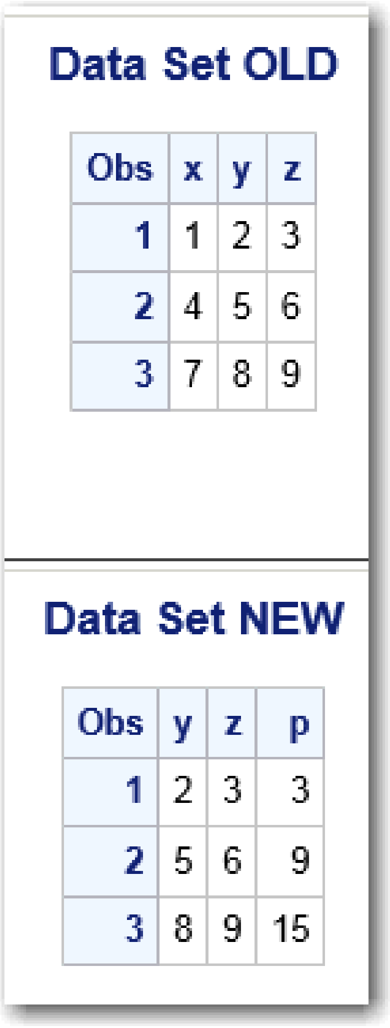
```
data new(drop=x);
  set old;
  p=x+y;
run;
```

$$\begin{tikzcd} \texttt{p=x+y;} \arrow[r] & \texttt{p=x+y;} \arrow[r] & \texttt{p=1+2;} \end{tikzcd}$$



Now the x value from the data set OLD is in the PDV and available for processing.

10. OUTPUT



Data Set OLD

Obs	x	y	z
1	1	2	3
2	4	5	6
3	7	8	9

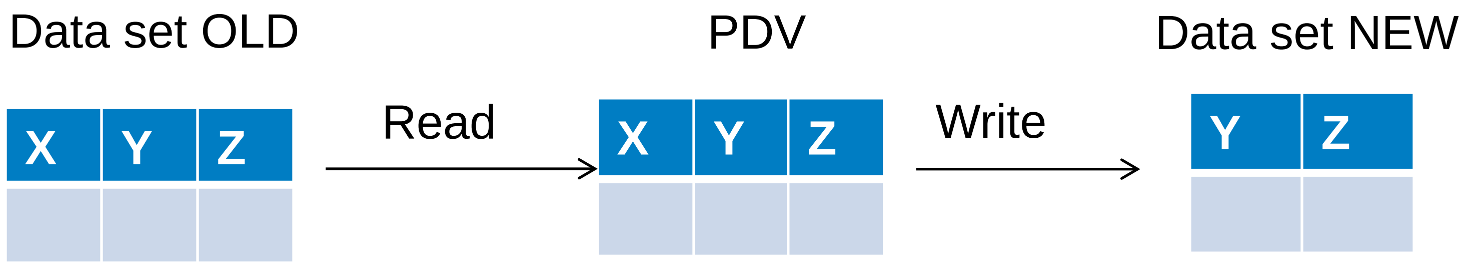
Data Set NEW

Obs	y	z	p
1	2	3	3
2	5	6	9
3	8	9	15

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

You may also use the DROP statement, which affects the output data set.

```
data new;
  drop x;
  set old;
run;
```

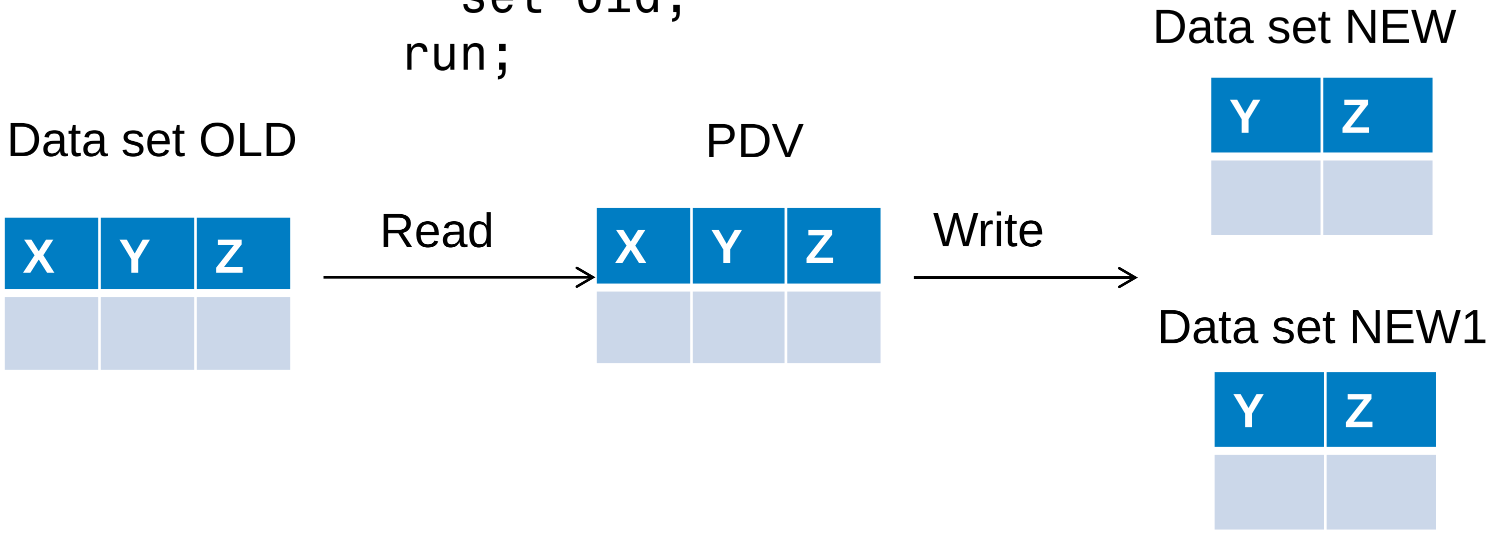


Note: The DROP statement affects ALL output data sets. The DROP= data set option affects only the data set it immediately follows.

10. SELECT ONLY THE COLUMNS YOU NEED WHEN WORKING WITH SAS DATA

You may also use the DROP statement, which affects the output data set.

```
data new new1;
  drop x;
  set old;
run;
```



Note: The DROP statement affects ALL output data sets. The DROP= data set option affects only the data set it immediately follows.

11. SELECT ONLY THE ROWS YOU NEED WHEN WORKING WITH SAS DATA

Do this:

```
data new;
   infile ‘old.dat’;
   if city=‘CLEVELAND';
   more statements here;
run;
```

Not this:

```
data new;
   infile ‘old.dat’;
   more statements here;
run;
```

12. CONSIDER THE POSITION OF THE SUBSETTING IF

Do this:

```
data new;
   infile ‘old.dat’;
   if city=‘CLEVELAND';
   more statements here;
run;
```

Not this:

```
data new;
   infile ‘old.dat’;
   more statements here;
   if city=‘CLEVELAND';
run;
```

Subset as soon as you have all necessary values in order to prevent unnecessary creation of variables and additional processing.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

Instead of this:

```
data new;
   set old;
   if condition;
   more statements here;
run;
```

Try this:

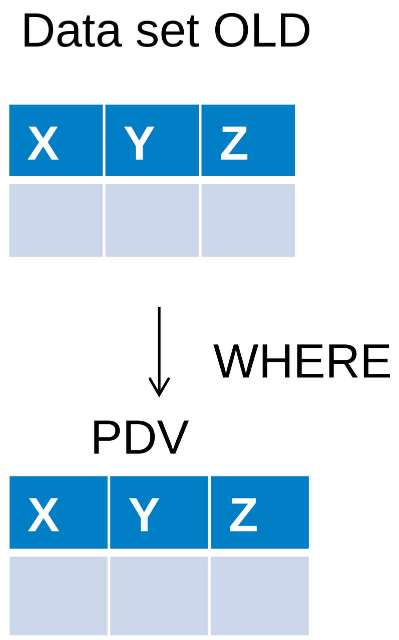
```
data new;
   set old;
   where condition;
   more statements here;
run;
```

Added efficiency: when using SAS/Access engines, SAS attempts to send the WHERE clause to the RDBMS for evaluation rather than to SAS; With the IF statement, SAS must do the processing.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

The WHERE statement is a pre-processor. It subsets data before it is loaded into the PDV.

```
data new;
  set old;
  where x > 100;
run;
```

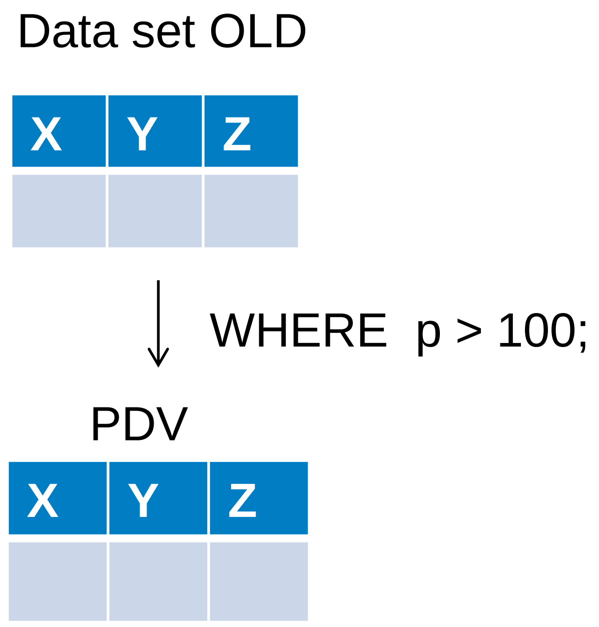


Since the WHERE statement subsets before loading data into the PDV, it expects to read SAS data.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

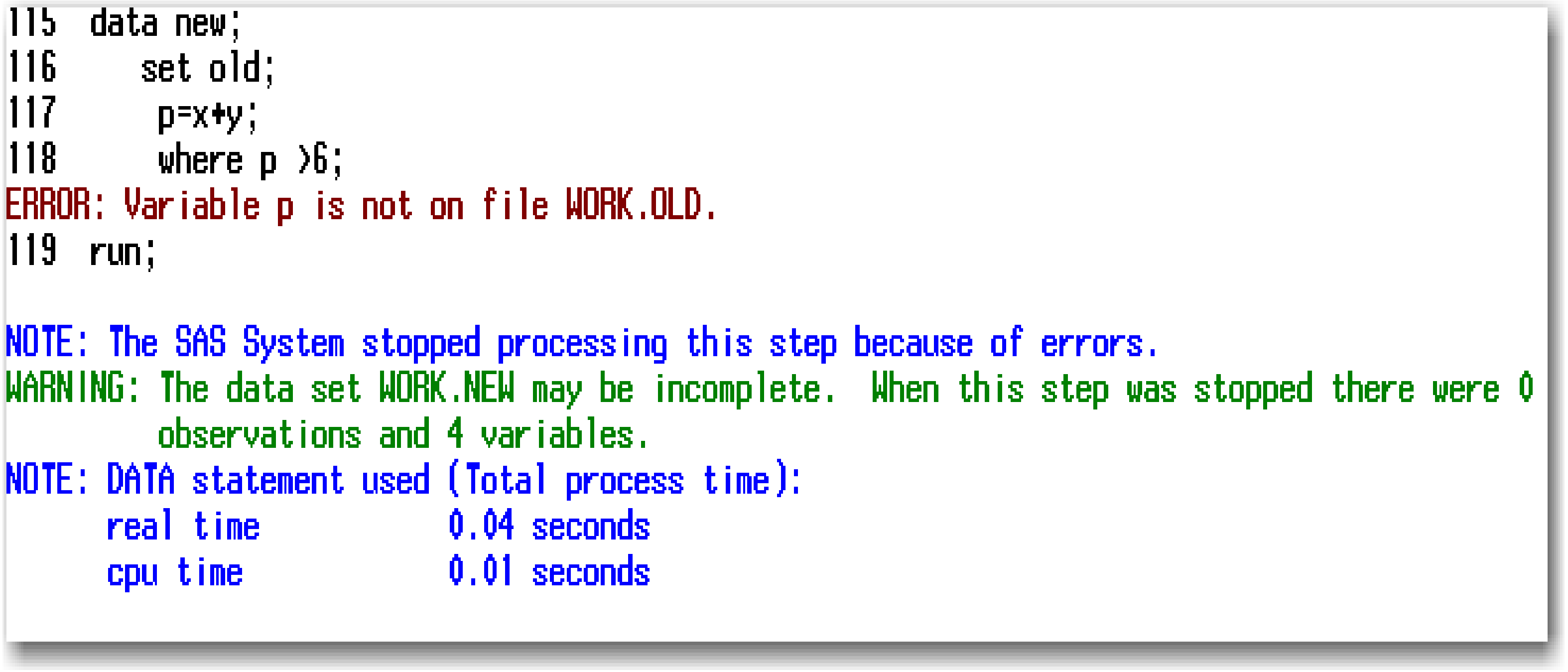
Consider the following SAS code.

```
data new;
  set old;
  p=x+y;
  where p > 100;
run;
```



The variable P does not exist in the input data set OLD. Since WHERE is a preprocessor, it can only ‘understand’ data that is stored in the SAS data set OLD. This code yields an error message.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF



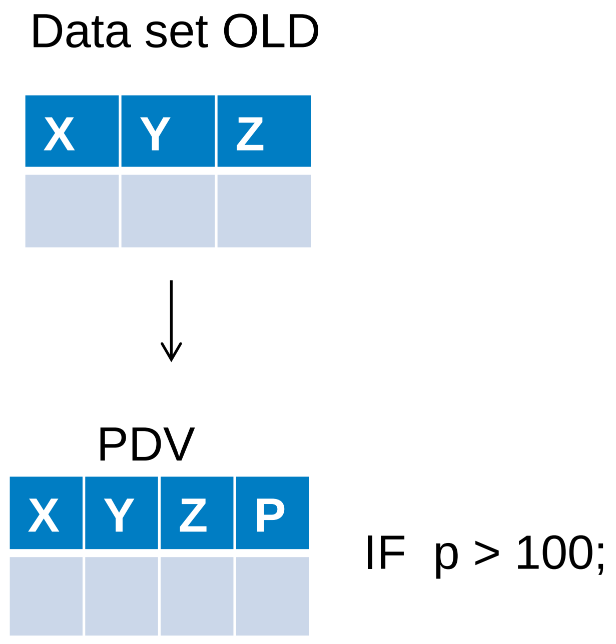
```
115  data new;
116     set old;
117      p=x+y;
118      where p >6;
ERROR: Variable p is not on file WORK.OLD.
119  run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.NEW may be incomplete.  When this step was stopped there were 0
         observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds
```

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

If you need to subset based on a calculated variable, you may choose to use the subsetting IF.

```
data new;
  set old;
  p=x+y;
  if p > 100;
run;
```

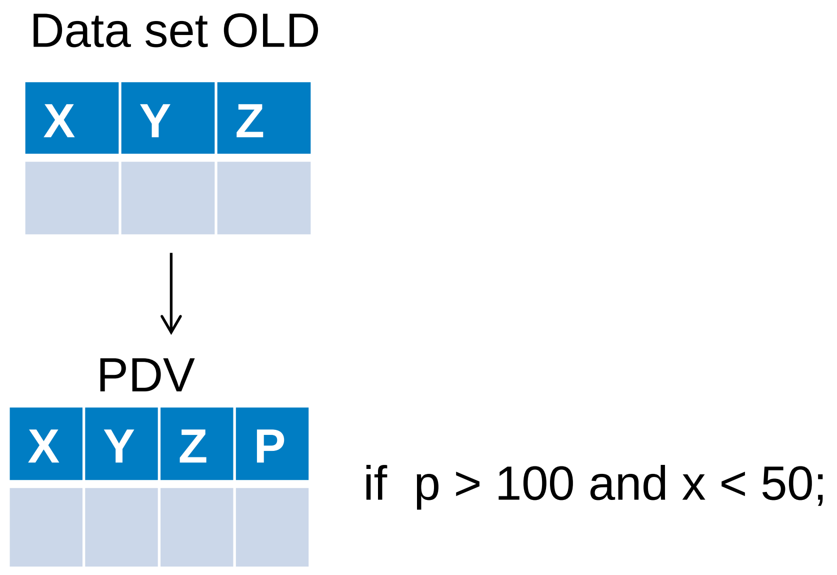


The subsetting IF subsets based on values that are in the PDV. It does not preprocess the data.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

When conditions permit, you may choose to use a combination of WHERE and subsetting IF. Consider the following code.

```
data new;
  set old;
  p=x+y;
  if p > 100 and x < 50;
run;
```

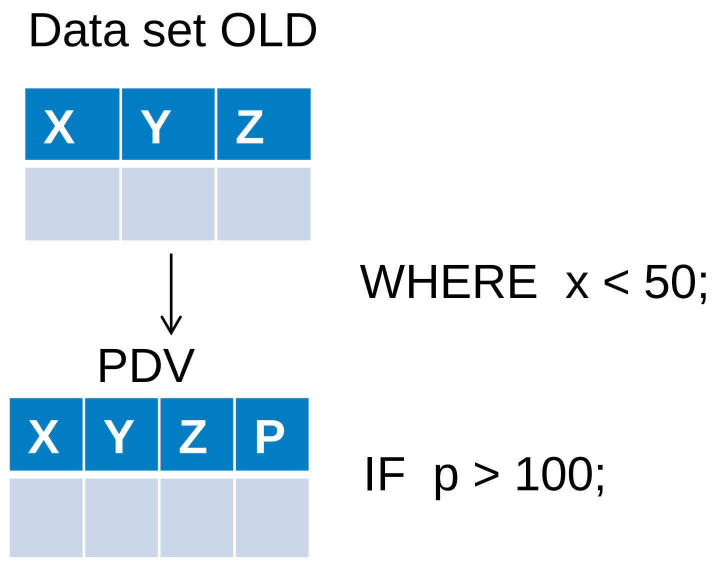


This code can be rewritten to use both WHERE and IF.

13. IF YOU ARE READING SAS DATA, USE WHERE INSTEAD OF SUBSETTING IF

Consider the following version of the code.

```
data new;
   set old;
   p=x+y;
   where x < 50;
   if p > 100;
run;
```



Advantage: The WHERE statement can still be used to subset appropriate information, reducing the amount of information subsequently loaded into the PDV.

14. CONSIDER DECLARING VARIABLES AS CHARACTER WHEN THERE IS A STORAGE SAVINGS

Consider Employee ID values similar to the following:

```
1015
2034
5543
6793
...
```

Compare:

```
data new;
  input ID 1-4;
```

- ID is numeric requiring 8 bytes of storage

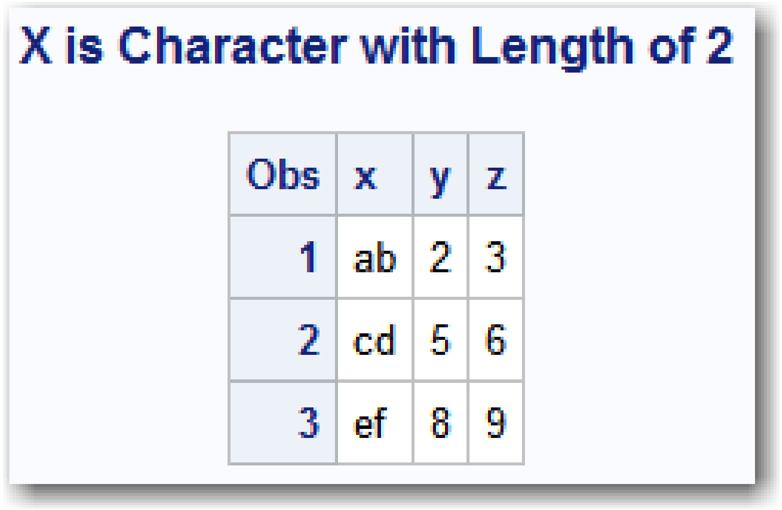
```
data new;
  input ID $ 1-4;
```

- ID is character requiring 4 bytes of storage

A savings of 4 bytes per observation adds up when dealing with large data!

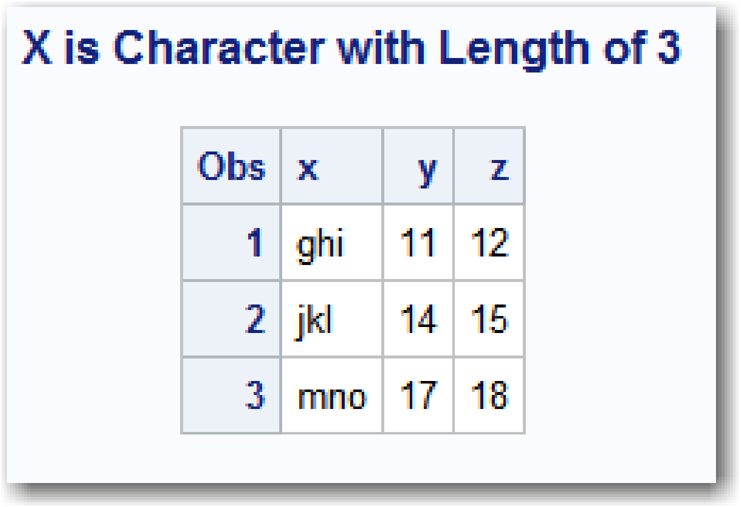
14A. USE THE LENGTH STATEMENT TO DECLARE CHARACTER VARIABLES

Consider the following data sets:



X is Character with Length of 2

Obs	x	y	z
1	ab	2	3
2	cd	5	6
3	ef	8	9

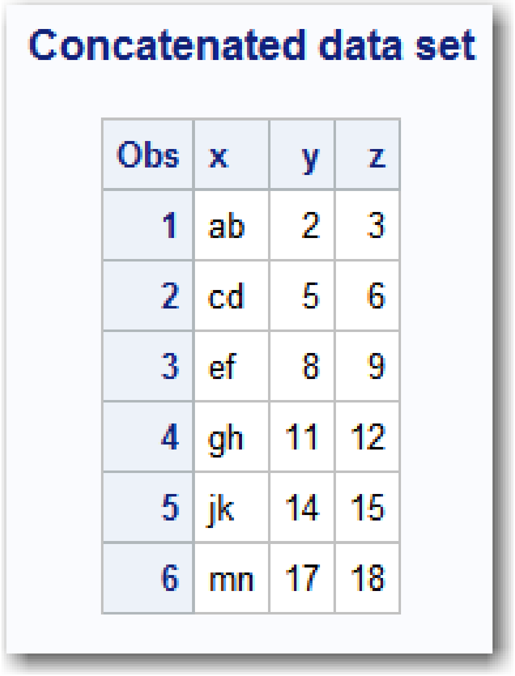


X is Character with Length of 3

Obs	x	y	z
1	ghi	11	12
2	jkl	14	15
3	mno	17	18

14A. USE THE LENGTH STATEMENT TO DECLARE CHARACTER VARIABLES

```
data concatenate;
  set old old2;
run;
```

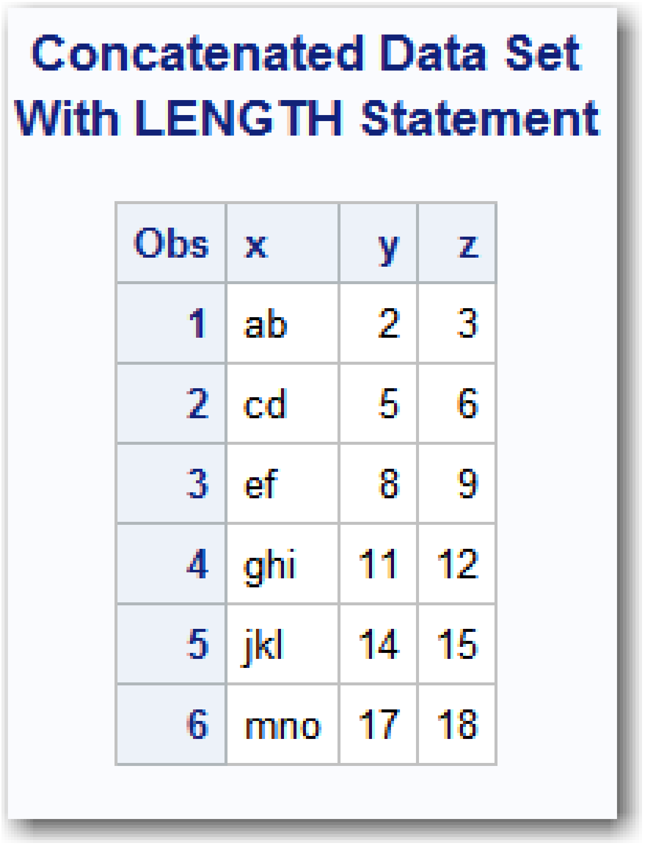

Concatenated data set

Obs	x	y	z
1	ab	2	3
2	cd	5	6
3	ef	8	9
4	gh	11	12
5	jk	14	15
6	mn	17	18

The code will produce truncated values for X. The value of X is established at compile time based on the attributes found the first time it is encountered. In this case, the attributes in the data set OLD are used because it is listed first on the SET statement.

14A. USE THE LENGTH STATEMENT TO DECLARE CHARACTER VARIABLES

```
data concatenate;
  length x $ 3;
  set old old2;
run;
```



Concatenated Data Set
With LENGTH Statement

Obs	x	y	z
1	ab	2	3
2	cd	5	6
3	ef	8	9
4	ghi	11	12
5	jkl	14	15
6	mno	17	18

15. CONSIDER ADDING INDEXES TO YOUR DATA IF YOU WILL BE FILTERING IT FREQUENTLY

What is an index?

An index is an optional file that you can create for a SAS data set in order to provide direct access to specific observations.

In other words, an index enables you to locate an observation by value.

15. WHAT IS AN INDEX?

The index file has the same name as its associated data file, and a member type of INDEX.

Indexed SAS Data Set

Obs	ID	Name
1	1001	Dunn
2	1002	Avery
3	1003	Brown
4	1004	Avery
5	1005	Craig

Index File

Value	Record Identifier
Avery	2, 4
Brown	3, 22, 43
Craig	5, 50
Dunn	1

Note: SAS automatically updates the index file as changes are made to the data.

15. WHEN TO USE INDEXES?

Index guidelines:

- Indexes perform best when they retrieve 15% or fewer rows in a table/data set.
- Indexes are not usually useful if they are based on uniformly distributed, low cardinality columns. (Male & Female example)
- Do not create indexes on small tables. Sequential access is faster.
- Minimize the number of indexes in order to reduce disk storage and update costs.

15. CONSIDER ADDING INDEXES TO YOUR DATA IF YOU WILL BE FILTERING IT FREQUENTLY

- Indexes can be created on the DATA statement, with PROC SQL, with PROC DATASETS, and in other ways.
- Indexes can be simple or composite.
- Under the right circumstances, indexes can decrease processing time.
- However, indexes take up space!
- Carefully consider whether an index makes sense in the specific situation.

Added efficiency: sort the data in ascending order on the key variable before indexing the data file.

15. METHODS FOR CREATING INDEXES

DATA step:

```
data finances(index=(stock /unique));
 more statements here
run;
```

PROC SQL:

```
proc sql;
     create unique index empnum
               on employee (empnum);

or
proc sql;
     create index names on
             employee(lastname, frstname);
```

15. METHODS FOR CREATING INDEXES

PROC DATASETS:

```
proc datasets library=mylib;
   modify my_dataset;
      index create empnum / unique;
      index create names=(lastname frstname);
run;
```

15. CONTROLLING INDEX USAGE

You can control index usage for WHERE processing with the DATA set options IDXWHERE and IDXNAME.

```
data mydata.empnew;
  set mydata.employee (idxwhere=yes);
     where empnum < 2000;
```

IDXWHERE=YES tells SAS to decide which index is the best for optimizing a WHERE expression, disregarding the possibility that a sequential search of the data file might be more resource efficient.

15. CONTROLLING INDEX USAGE

```
data mydata.empnew;
   set mydata.employee (idxname=empnum);
  where empnum < 2000;
```

The IDXNAME= data set option directs SAS to use a specific index in order to satisfy the conditions of a WHERE expression.

16. USE CEDA WISELY

What is CEDA?

- CEDA stands for Cross-Environment Data Access, and refers to a component of the SAS data architecture that allows SAS to access physical data using a platform other than the one used to create the data.
- CEDA describes the capability to recognize non-native data and the components that arrange for cross-architecture data translation on the fly.

16. USE CEDA WISELY

- Reading SAS 9.2 or earlier data sets in SAS 9.3 results in a translation process using CEDA (cross-environment data access)
- Because the BASE engine translates the data as the data is read, multiple procedures require SAS to read and translate the data multiple times. In this way, the translation could affect system performance.
- “Convert” SAS data sets by using PROC MIGRATE or other techniques.

WHEN YOU ARE DEVELOPING CODE

Tips to save time and create efficient code.

17. TEST YOUR PROGRAMS WITH THE OBS= OPTION

```
data complicated_program;
  set sample_data(obs=50);
  many, many, many more statements here;
run;
```

This technique may not adequately test all conditions, but will confirm the correctness of the overall program logic – and save time and computer resources!

17A. TEST YOUR PROGRAMS WITH THE PUT STATEMENT

```
data complicated_program;
  set sample_data;
  if condition then do;
      put ‘write value here’ value;
      other statements to execute;
     end;
run;
```

This technique allows you to test certain coding logic to determine if conditions are met as well as variable values.

18. BENCHMARK PRODUCTION JOBS

Recommendations for benchmarking include:

- Benchmark your programs in separate SAS sessions
- Run each program multiple times and average the performance statistics.
- Use realistic data for tests.
- Elapsed time should not be used for benchmarking.

19. MAKE THINGS EASIER FOR YOURSELF: EFFICIENCY ALSO MEANS WORKING SMARTER!

- Be “GREEN” – save code and reuse it later!
- Collaborate with your co-workers to share tips and suggestions
- Meet regularly to share ideas
- Some ways SAS code fosters reusability:
  - » Macro library
  - » Stored processes
  - » User-written functions and procedures.

RESOURCES:

- http://support.sas.com
- SAS Training
- Local and regional users groups
- Your Customer Account Executive
- Your co-workers and peers!

It's where you'll find the latest information on:

- New training courses and services
- Special offers and discounts
- The latest course schedules
- New training locations
- Events and conferences
- SAS certification news
- And, much more.

Everything you need – in one place.
Visit and bookmark it today.

THANK YOU!