

```

*****
*****;
***
***;
***   BASE SAS Tips and Tricks
***;
***
***;
***   Amalgamated by: Rob Wilson (from many different sources)
***;
***
***;
***   Presented: Victoria User Group on Oct 6th, 2009
***;
***                   Vancouver User Group on Oct 7th, 2009
***;
***
***;
*****
*****;

/* start with shortcuts */
/* create shortcuts using the Tools -> Add Abbreviations option:*/
/* SAS note on support.sas.com for instructions and available operating
systems: http://support.sas.com/kb/32/774.html */
/* Additional online instructions:
http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/eeshortcuts.h
tm */

/* my abbreviation "d" gives: */
data ;
  set ;
run;

data ;
  set ;
run;

* Please note;;
/* keyboard abbreviations are not available in Unix but function keys
can be assigned in Unix */
/* specific placement of cursor is available using recorded macros and
commands */
/* Enhanced editor macros, abbreviations and shortcut keys are stored in
the windows registry,
  but can also be backed-up and shared by saving them to external
files.
  SUGI paper on abbreviations can be found here:Notes can be found
herehttp://www2.sas.com/proceedings/sugi31/237-31.pdf*/

*=====
=====;
*=====
=====;

/* what do i have licensed on my machine */
* Proc setinit. tells you what you have currently valid licenses and
site number, and defined products in your session. ;

```

```

proc setinit;
run ;

*=====
=====;
*=====
=====;

/* Customizing a toolbar */
/*
To customize the toolbar just right click on it, and choose customize.
Remember you have different toolbars for different windows (log,
enhanced editor, etc.).

I have found the following tools very useful to have on the toolbar for
the log window.
I have listed the command to enter when defining each tool, and
suggested some icons.
- find "ERROR:" icase all: finds first "ERROR:" (some kind of
magnifying glass)
- rfind: finds next match (use a right or down icon) Adding these icons
means you can look for errors
and then find the next error(s). Give it a try!
*/
*=====
=====;
*=====
=====;

/* progress indicator for long processes using the Window statement */
/* window statement is available in Unix Tru64 */
%macro progress(every);
    window progress irow=4 rows=7 columns=40
    #1 @6 'Processing record: ' _n_ persist=yes;
    if mod(_n_,&every)=0 then display progress noinput;
%mend progress;

data sample01;
    %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
    infile 'C:\Data\SAS\Victoria_UG\sample.txt' delimiter='09'x MISSOVER
DSD lrecl=32767 firstobs=2 ;
/* please note that this file was not provided as it was too big to send
via email */
    informat x best32. ;
    informat y $50. ;
    informat z $50. ;
    format x best12. ;
    format y $50. ;
    format z $50. ;

    input
        x
        y $
        z $
    ;
    if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection
macro variable */

```

```

        %progress(50000);

run;

/* you can also use WINDOW and DISPLAY statements to enter user
parameters: http://support.sas.com/kb/37/055.html */
data ;
  set ;
run;
*=====
=====;
*=====
=====;

/* creating sounds with SAS */
* if there is an error;
data sample02;
  input x;
  cards;
  1
  4
  7
  0
  ;
run;

data sample03;
  set sample02;

  y = 5/x;

  if _error_= 1 then
    do;
      call sound(46,200);
    end;
run;

* when a program is finished;
data sample02;
  input x;
  cards;
  1
  4
  7
  0
  ;
run;

data _null_;
  x 'c:\data\rooster1.wav';
run;

/* Old MacDonald Had a Farm */
data _null_;
call sleep(1,1);
run;
%let pc=1.25;
%macro df3(note,octave,length);
select(&note.);

```

```

when('A') call sound(55*(2**&octave.),&length.*160*&pc.);
when('A#') call sound(58*(2**&octave.),&length.*160*&pc.);
when('Bb') call sound(58*(2**&octave.),&length.*160*&pc.);
when('B') call sound(62*(2**&octave.),&length.*160*&pc.);
when('C') call sound(65*(2**&octave.),&length.*160*&pc.);
when('C#') call sound(69*(2**&octave.),&length.*160*&pc.);
when('Db') call sound(69*(2**&octave.),&length.*160*&pc.);
when('D') call sound(73.5*(2**&octave.),&length.*160*&pc.);
when('D#') call sound(73.5*(2**&octave.),&length.*160*&pc.);
when('Eb') call sound(78*(2**&octave.),&length.*160*&pc.);
when('E') call sound(82*(2**&octave.),&length.*160*&pc.);
when('F') call sound(87*(2**&octave.),&length.*160*&pc.);
when('F#') call sound(92.5*(2**&octave.),&length.*160*&pc.);
when('Gb') call sound(92.5*(2**&octave.),&length.*160*&pc.);
when('G') call sound(98*(2**&octave.),&length.*160*&pc.);
when('G#') call sound(104*(2**&octave.),&length.*160*&pc.);
when('Ab') call sound(104*(2**&octave.),&length.*160*&pc.);
when('R') call sleep((&length./3)*&pc.,1);
otherwise;
end;
%mend;
data _null_;
do i=1 to 2;
%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,1);
%df3('G',2,1);
%df3('A',3,1);
%df3('A',3,1);
%df3('G',2,2);
%df3('E',3,1);
%df3('E',3,1);
%df3('D',3,1);
%df3('D',3,1);
%df3('C',3,2);
if i=1 then do;
%df3('R',1,2);
%df3('G',2,2);
end;
end;
%df3('G',2,.5);
%df3('G',2,.5);
%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,1);
%df3('G',2,.5);
%df3('G',2,.5);
%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,2);
%df3('C',3,.5);
%df3('C',3,.5);
%df3('C',3,1);
%df3('C',3,.5);
%df3('C',3,.5);
%df3('C',3,1);
%df3('C',3,.5);
%df3('C',3,.5);
%df3('C',3,.5);
%df3('C',3,.5);

```

```

%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,1);
%df3('C',3,1);
%df3('G',2,1);
%df3('A',3,1);
%df3('A',3,1);
%df3('G',2,2);
%df3('E',3,1);
%df3('E',3,1);
%df3('D',3,1);
%df3('D',3,1);
%df3('C',3,3);
run;

```

```

*=====
=====;
*=====
=====;

```

/* misc tip

More than once I've found myself tediously typing out long strings of variables, so I set out on a mission to find as many ways as possible to avoid typing out those long strings. These are the possible solutions I found:

1) the dash (-): Using a dash between two variable names will create an ordered list of variables.

```

So           KEEP VAR1-VAR4;
is equivalent to KEEP VAR1 VAR2 VAR3 VAR4;

```

Note, if one of the variables expected in the ordered list does not exist a WARNING will be issued to the log. However, if using the dash when defining an array, the missing variable will be created.

2) the double-dash (--): Using a double dash between two variable names will create a list of variables as they are ordered in the dataset.

So if I have the following variables in a dataset in this order, APPLE, ORANGE, BANANA, GRAPE,

```

then           KEEP APPLE -- BANANA;
is equivalent to KEEP APPLE ORANGE BANANA;

```

3) the colon (:): Using a colon after a string of letters will create a list of all variables prefixed with those letters.

That is var: will create a list of all variables in the dataset that begin with those 3 letters.

If I have a dataset with variables named TESTIN TESTOUT TESTRES TESTNAME VDATE and ID.

```

Then           KEEP TEST:;
is equivalent to KEEP TESTIN TESTOUT TESTRES TESTNAME;

```

The colon cannot be used in front of a variable to find variables ending

in a pattern of characters.

4) the `_ALL_`: Using the `_ALL_` will create a list of all the variables in the dataset.

So if I have a dataset with variables VAR1 VAR2 VAR3 and VAR4.

```
Then          KEEP _ALL_;  
is equivalent to KEEP VAR1 VAR2 VAR3 VAR4;
```

This one can be most useful with a proc sort nodupkey to remove records which are exact matches on all variables.

```
proc sort data = sample nodupkey;  
  by _all_;  
run;
```

Also can be used to strip off existing formats:

```
data sample;  
  set sample;  
  format _all_;  
run;
```

5) the `_CHARACTER_`: Using `_CHARACTER_` creates a list of all the character variables in the dataset. If I have a dataset with two character variables CVAR1 and CVAR2 and two numeric variables NVAR3 NVAR4,

```
then          KEEP _CHARACTER_;  
is equivalent to KEEP CVAR1 CVAR2;
```

6) the `_NUMERIC_`: Using `_NUMERIC_` creates a list of all the numeric variables in the dataset.

So in the same dataset from #5,

```
then          KEEP _NUMERIC_;  
is equivalent to KEEP NVAR1 NVAR2;
```

Both `_CHARACTER_` and `_NUMERIC_` can be useful when creating arrays, since all the variables in the array must be the same type.

Also handy for doing univariates or frequencies on all the numeric or character variables, respectively in a dataset.

*/

```
*=====
```

```
/* reading files from a webpage */  
FILENAME mine URL 'http://support.sas.com/documentation/index.html';  
data webpage;  
  infile mine length= len;  
  input record $varying200.len;  
  
  put record $varying200.len;
```

```

    if _n_ = 15 THEN stop;

run;

*=====
*=====;
*=====
*=====;

/* creating your own functions */

* count days from randomization to event;
proc fcmp outlib = sasuser.funcs.mine;
    function count_day(rand_date, event_date);

        if event_date < rand_date then return (.);
        else return (event_date - rand_date);

    endsub;
quit;

/*
Testing the functions can also be done in PROC FCMP using PUT statements
• We need to first tell SAS where our function library of compiled
functions is using an OPTIONS statement
• Test results from PROC FCMP go to the list file by default
*/

options cmplib=(sasuser.funcs);

/* test the COUNT_DAY */
proc fcmp;
    out2 = count_day('15Feb2006'd,'28Feb2007'd);
    put "Testing Function Call !!!";
    put "count_day(Feb 15, 2006 - Feb 28, 2007) returns:" out2;
quit;

* now you can have a function called BEER!;
* 1 = domestic
* 2 = import;

proc fcmp outlib = sasuser.funcs.mine;
    function beer(type, amount);

        if type = 1 then return (amount * 5);
        else return (amount * 7.5);

    endsub;
quit;

options cmplib=(sasuser.funcs);

/* test BEER */
proc fcmp;
    wallet = beer(1,5);
    put "Testing Function Beer !!!";

```

```
    put "Tonight will cost me: $" wallet;
quit;
```

```
proc fcmp;
    wallet = beer(2,5);
    put "Testing Function Beer !!!";
    put "Tomorrow will cost me: $" wallet;
quit;
```

```
*=====
=====;
*=====
=====;
```

```
/* neat function */
* COMPARE(string1, string2, optional modifiers)
- returns the location of the first difference between the two strings
OR 0 if they are the same;
```

```
data temp;
    s1="hypothesis";
    s2="hypoThesis";
    d1=compare(s1,s2);
    d2=compare(s1,s2,"i");
run;
```

```
*...will return "5" for d1 and "0" for d2;
```

```
*=====
=====;
*=====
=====;
```

```
/* misc tip (not SAS related)
/* The following tip comes from http://spence.nu/blog/?m=200707 (a site
which has since been taken down) and seemed useful
    for those running SAS under Windows. It means you can easily clean up
your memory without the need for costly tools.
```

If you run a Windows computer you'll know like many others that after a while your system will no doubt start running slow. Most people will restart their computer to remove any idle processes. But if there's a simpler way, why restart every time Windows decides it doesn't like you today?

1. Right click on an empty spot on your desktop and select New --> Shortcut.
2. Type %windir%\system32\rundll32.exe advapi32.dll,ProcessIdleTasks in the box.
3. Click Next.
4. Give your shortcut a nice name like "Clear Memory".
5. Click Finish and you're done.

Now whenever your computer starts running slow click this shortcut to clear out your memory and get your computer running at a normal pace again.

More information can be found here ...
<http://www.tek-tips.com/faqs.cfm?fid=4518>

or
<http://www.techzonez.com/forums/archive/index.php/t-945.html>
or
<http://dev.remotenetworktechnology.com/cmd/rundll32.htm>

```
*=====
=====;
*=====
=====;
/* ASCII line breaks hidden in character variables
/* typically found when trying to merge and datasets and an error is
driving you crazy
```

Example:

- every term in SETA is supposed to have a matching record in SETB.
- for some reason the merge is 'missing' on several of the observations.
- the terms that aren't matching in the merge are in both SETA and SETA.
- and they looked identical!
- so why isn't the merge matching!!!

One way to find hidden line breaks

- use the PUT statement to print contents of a variable in the log. if there is an unexpected line break in the output, most likely an ASCII line break in the variable
- particularly common in Excel files
- sometimes you'll actually see a little square when you view the SAS dataset

Solution!:

- Very simple
- Use the TRANWRD function to replace the hidden ASCII characters with a blank:

```
term=tranwrd(term,'0A'x,'');
term=tranwrd(term,'0D'x,'');
```

Those are zeros, not big o's.

Note that there are two types of ASCII line breaks. One is '0A'x and the other is '0D'x.

You can use one or the other, whichever solves your problem. Generally use both, to make sure you catch whatever may be there.

Whenever you have a problem with merges, invisible ASCII characters are the first suspect.

```
*/
```

```
*=====
=====;
*=====
=====;
```

```
/* misc tip - FONTS
```

Ever waste time debugging a program, only to discover that you've typed a zero (0) instead of an 'O'?

Or an upper case 'I', rather than a lower case 'l'?

Try switching your editor font (Tools-Options-Enhanced Editor) to SAS Monospace.

```
*/
```

```

*=====
=====;
*=====
=====;

/* misc tip - Writing times in logs

Ever wonder how long a program runs?

This program contains sample code which demonstrates how to write a
program's start time, end time, and processing time
in minutes to the SAS Log.

Customize it to suit your needs.
If you start off your %put statement with 'NOTE:' it will appear in the
log as if it were a NOTE: generated by the SAS system directly.
*****
*****;
* Write start time to the SAS log;
*****
*****;*/
%let startTime=%sysfunc(datetime(),DATETIME16.);
%put "NOTE: This program started processing at" &startTime.;

/* Which writes NOTE: This program started processing at
DDMONYY:HH:MM:SS to the log. */

/* [Your SAS program goes here.] */

*****
*****;
* Write end time to the SAS log;
*****
*****;
%let endTime=%sysfunc(datetime(),DATETIME16.);
data _null_;
timeLapse = intck('minutes',"&startTime."dt,"&endTime."dt);
call symputx('totalProcesTime',timeLapse);
run;
%put NOTE: This program completed processing at &endTime. and took
&totalProcesTime. minutes to complete.;

/* Which writes NOTE: This program completed processing at
23APR09:17:53:52 and took 124 minutes to complete. to the log. */
*/

*=====
=====;
*=====
=====;

/* checking properly formatted emails */
data _null_;
retain re;
length email $ 150;

if _N_ = 1 then
do;
/* Simple regex for most common email addresses */

```

```

    regexp = "/^[\\w.%+-]+@[\\w.-]+\\. [a-z]{2,4}\\s*$/i";
    re = prxparse(regexp);
    if missing(re) then
        do;
            putlog "ERROR: Invalid regexp " regexp;
            stop;
        end;
    end;

input email;

    if ^prxmatch(re, email) then
        putlog "invalid: " email;
    else
        putlog "    ok: " email;
datalines;
not.an.email
not@an.email.either
john.smith@uwa.edu.AU
frank@hotmail.com
frank@home.com,frank.work.com
;

*=====
=====;
*=====
=====;

/* Reorder variables' position in a SAS data set?*/

* You can add either Length or Retain statement before a Set statement
in a data step.

For example;
data a;
input a1 b1 c1 a2 b2 c2 a3 b3 c3;
cards;
1 2 3 4 5 6 7 8 9
;

data b;
length a1-a3 3. b1-b3 3. c1-c3 3. ;
set a;
run;

data c;
retain a1-a3 b1-b3 c1-c3;
set a;
run;

*=====
=====;
*=====
=====;

/* Proper case */
*PROPCASE function in SAS 9.1. For example;

data proper;
input name $40.;

```

```

name = propcase(name);
datalines;
jeff Rust 123 smith st
Amy lee
;
proc print; run;

*=====
=====;
*=====
=====;

/* How to generate a directory file with total observations, total
variables, file size and the last time the file was modified
for a defined library name? */

* Use proc datasets to check information of total observations, total
variables, file size and the last time the file was modified
for a defined "library" from log window;

proc datasets library=work mt=data details;
run;
quit;

* Use ODS to save them to a new data set;

ods output members=dir_work;
proc datasets library=work mt=data details;
run;
quit;
ods output close;

*=====
=====;
*=====
=====;

/* some neat stuff */
* links in output;
* embedded pix;

ods html body="odstab11.htm" file="c:\temp_html.html";
ods listing close;
title '';
footnote <A HREF="http://www.SAS.com">SAS.com</A>;

data tabulate;
input dept acct qtr mon expense @@;
cards;
1 1345 1 1 12980 1 1674 1 3 13135 3 4138 1 1 29930
1 1345 1 1 9475 1 1674 1 3 21672 3 4138 1 2 22530
1 1345 1 1 15633 1 1674 1 3 3847 3 4138 1 2 16446
1 1345 1 2 14009 1 1674 1 3 2808 3 4138 1 2 27135
1 1345 1 2 10226 1 1674 1 3 4633 3 4138 1 3 24399
1 1345 1 2 16872 2 2134 1 1 34520 3 4138 1 3 17811
1 1345 1 2 17800 2 2134 1 1 25199 3 4138 1 3 29388
1 1345 1 2 12994 2 2134 1 1 41578 3 4138 1 3 16592
1 1345 1 2 21440 2 2134 1 2 26560 3 4138 1 3 12112
1 1345 1 3 35300 2 2134 1 2 19388 3 4138 1 3 19984

```

```

1 1345 1 3 25769 2 2134 1 2 31990 3 4279 1 1 9984
1 1345 1 3 42518 2 2134 1 3 24399 3 4279 1 1 7288
1 1578 1 1 8000 2 2134 1 3 17811 3 4279 1 1 12025
1 1578 1 1 5840 2 2134 1 3 29388 3 4279 1 2 14209
1 1578 1 1 9636 2 2403 1 1 25464 3 4279 1 2 10372
1 1578 1 2 7900 2 2403 1 1 18588 3 4279 1 2 17113
1 1578 1 2 5767 2 2403 1 1 30670 3 4279 1 3 13500
1 1578 1 2 9515 2 2403 1 2 15494 3 4279 1 3 9855
1 1578 1 3 4500 2 2403 1 2 11310 3 4279 1 3 16260
1 1578 1 3 3285 2 2403 1 2 18661 3 4290 1 1 10948
1 1578 1 3 5420 2 2403 1 2 1482 3 4290 1 1 7992
1 1674 1 1 11950 2 2403 1 2 1081 3 4290 1 1 13186
1 1674 1 1 8723 2 2403 1 2 1783 3 4290 1 2 14539
1 1674 1 1 14392 2 2403 1 3 10009 3 4290 1 2 10613
1 1674 1 2 13534 2 2403 1 3 7306 3 4290 1 2 17511
1 1674 1 2 9879 2 2403 1 3 12054 3 4290 1 3 11459
1 1674 1 2 16300 3 4138 1 1 24850 3 4290 1 3 8365
1 1674 1 3 17994 3 4138 1 1 18140 3 4290 1 3 13802
;

run;

proc format;
  value qtrfmt 1 = 'FIRST QUARTER'
              2 = 'SECOND QUARTER'
              3 = 'THIRD QUARTER'
              4 = 'FOURTH QUARTER';

  value monfmt 1 = 'January'
              2 = 'February'
              3 = 'March'
              4 = 'April'
              5 = 'May'
              6 = 'June'
              7 = 'July'
              8 = 'August'
              9 = 'September'
             10 = 'October'
             11 = 'November'
             12 = 'December';

  value dept 1 = 'Accounting'
            2 = 'Human Resources'
            3 = 'Systems';

run;

proc tabulate format=dollar11.2;

  /* Give headings a purple foreground. */

  class mon qtr acct dept / style={foreground=purple};
  classlev mon qtr acct dept / style={foreground=purple
                                     font_style=italic};
  var expense / style={foreground=purple};

  format qtr qtrfmt.;
  format mon monfmt.;
  format dept dept.;
  label expense = "Expenses";

```

```

/* Left-justify and italicize row total heading */
table dept=' ' all= 'Totals'

/* Highlight row totals with a red background. */
    *{style={background=red}},

/* Italicize column total heading. */
    (mon=' ' all={label="First Quarter"
                  style={foreground=purple}}

/* Highlight column totals with a red background. */
    *{style={background=red}})
    *expense*sum=' ' /

/* Make table background to be green. */
    style={background=green}

/* Display a graphic image in the box above the row */
/* headings. */

/* Place your image here. */
    box={ style={ flyover="Place your image HERE"
                  preimage="C:\Documents and Settings\CANRAW\My
Documents\My Pictures\sprocket.jpg"
                  prehtml=''
                  posthtml=''}};
run;

/* All done, let us take a look. */

ods html close;

*=====
*=====;
*=====
*=====;

* zipping files;
x 'c:\progra~1\winzip\winzip32.exe -a -s"pass1234" c:\test.zip c:
\weir_using_csv.csv';
/* where pass1234 is the password */

*=====

```

```
=====;
*=====;
=====;
```

```
* PUT and INPUT Statements
```

```
- PUT function returns a numeric value as a character string
- INPUT function returns the value of the character value as a numeric
value;
```

```
*Example 1:;
```

```
data testin;
```

```
input sale $9.;
```

```
fmtsale=input(sale,comma9.);
```

```
datalines;
```

```
2,115,353
```

```
;
```

```
*Example 2: Using PUT and INPUT Functions
```

```
In this example, PUT returns a numeric value as a character string.
```

```
The value 122591 is assigned to the CHARDATE variable.
```

```
INPUT returns the value of the character string as a SAS date value
using a SAS date informat.
```

```
The value 11681 is stored in the SASDATE variable. ;
```

```
data put_sample;
```

```
format sasdate2 date9.;
```

```
numdate=122591;
```

```
chardate=put(numdate,z6.);
```

```
sasdate=input(chardate,mmddy6.);
```

```
sasdate2 = sasdate;
```

```
run;
```

```
*-----Using the PICTURE statement to create new date
```

```
formats-----;
```

```
proc format;
```

```
/* 0m=month ; 0d=day of month ; y=year ; Y=century+year */
```

```
picture mmddy_a
```

```
low-high = '%0m%0d%y' (datatype=date);
```

```
picture mmddy_yy_b
```

```
low-high = '%0m%0d%Y' (datatype=date);
```

```
picture yyyyymmdd_c
```

```
low-high = '%Y%0m%0d' (datatype=date);
```

```
picture robdt
```

```
low-high = '%Y-Rob-%0m-Is-%0d-Awesome.' (datatype=date);
```

```
run;
```

```
data sample04;
```

```

day = today();

format day date9.;

monyear  = put(day,mmddy_a.);
monyear4 = put(day,mmddy_b.);
yearmon4 = put(day,yyyymmdd_c.);

robdate  = put(day,robdt.);

run;

```

```

*=====
*=====;
*=====
*=====;

```

* Methods for commenting out large chunks of code:

```

*1) * comment here ;    *(or %* comment here; *for use in a macro);
*2) /* comment here */;

```

```

/*
and now a THIRD way....

```

3) wrap the section to be commented in a macro and never call the macro. When we are modifying/debugging code we can comment using method 1:

```

*data test;
* set test1;
* var1=substr(var1a,2,34);
* var3=var2*var4/var5;
*run;

```

Which can be very time consuming if we need to comment out large sections of code.

Or using method 2:

```

/*
data test;
set test1;
var1=substr(var1a,2,34);
var3=var2*var4/var5;
run;
*/

```

* which is much less time consuming, but can be problematic if you already have /* */ commenting in the code.

For example, if we tried to comment out the following code by putting a /* before and a */ after, only the sections marked in green would actually be commented out.

```

/*
/*-----Step1: Finding baseline-----*/
proc sort data = raw.labs(where=(studyday lt 1))
    out = baseline(rename=(lab=b_lab));
    by patient studyday;
run;

```

```

/*-----Step2: Calculating Change-----*/
data change;
  set raw.labs
      baseline;
  by patient;

  chg=lab-b_lab;

run;
*/

```

* That would not achieve our purposes at all. We would have to go through and put in several sets of /* */ to wrap the code correctly.

Or we can use method 3:;

```

%macro skip;
/*-----Step1: Finding baseline-----*/
proc sort data = raw.labs(where=(studyday lt 1))
  out = baseline(rename=(lab=b_lab));
  by patient studyday;
run;

/*-----Step2: Calculating Change-----*/
data change;
  set raw.labs
      baseline;
  by patient;

  chg=lab-b_lab;

run;
%mend skip;

```

* This will comment out the whole chunk without regard to any of the existing comments.
 It doesn't turn a pretty green color in the PC SAS Enhanced Editor, but nonetheless it will no longer be executed when the program is run.

An additional advantage - when looking at programs with a viewer other than the Enhanced Editor (eg. SAS View or UNIX), the sections commented out with this last method are much clearer than sections commented out with multiple /**/ comments.

*/

/* Method 4:

To enclose code or text within a block comment, highlight the selected code and press Ctrl+/
 To remove a block comment, highlight the selected code and press Ctrl+Shift+/
 This works in the SAS enhanced editor.

```

*/

*=====
=====;
*=====
=====;

libname sasuser "C:\Documents and Settings\CANRAW\My Documents\My SAS
Files\9.1";
libname temp "C:\Data\SAS\Victoria_UG";
options fullstimer;

*****;
* Finding the number of observations in SAS dataset *;
*****;

*1 First up is the most basic and least efficient method: read the whole
data set and increment a counter a pick up its last value.
The END option allows you to find the last value of count without
recourse to FIRST.x/LAST.x logic.;

data count01;
set temp.sample NOBS=size;

call symput("count1",size);

run;
/* real time
cpu time
memory
*/
%put &count1;

*2 The next option is a more succinct SQL variation on the same idea.
The colon prefix denotes a macro variable whose value is
to be assigned in the SELECT statement. there should be no surprise
as to what the COUNT(*) does... ;

proc sql;
select count(*)
into :COUNT2
from temp.sample;
quit;
%put &count2;

/*
For details on why you shouldn't use NOBS, see:
<http://www2.sas.com/proceedings/sugi26/p095-26.pdf>

Basically, if you know that your data set is a native data set which has
never had observations deleted (e.g., never been edited interactively),
it's ok to use NOBS. If you need to know the number of observations in
an arbitrary data set (in a macro, for instance), you have to count
them.
*/

```

*3 Continuing the SQL theme, accessing the dictionary tables is another route to the same end and has the advantage of needing to access the actual data set in question. You may have an efficiency saving when you are testing large datasets but you are still reading some data here. ;

```
proc sql;
  select nobs
  into   :count3
  from   dictionary.tables
  where  libname eq 'TEMP'
  and    memname eq 'SAMPLE';
quit;
%put &count3;
```

*4 The most efficient way to do the trick is just to access the data set header. Here's the data step way to do it;;

```
data _null_;

  if 0 then set temp.sample nobs=size;

  call symputx("count4",size);

  stop;
run;
%put &count4;
```

* The IF/STOP logic stops the data set read in its tracks so that only the header is accessed, saving the time otherwise used to read the data from data set. Using the SYMPUTX routine avoids the need to explicitly code a numeric to character transformation. it's a SAS 9 feature, though.

*5 most succinct and efficient way of all: the use of macro and SCL functions.;

```
%macro count2;
%let dsid = %sysfunc(open(temp.sample));
%let num  = %sysfunc(attrn(&dsid,nlobs));
%if &DSID > 0 %then %let rc = %sysfunc(close(&dsid));
%mend;

%count2;
%put There are &num observations in the dataset;
```

* The first line opens the data set and the last one closes it. this is needed because you are not using data step or SCL and so could leave a data set open, causing problems later. The second line is what captures the number of observations from the header of the data set using the SCL ATTRN function called by %SYSFUNC.

```
*6 Last option;
PROC DATASETS LIBRARY=work NOLIST;
  CONTENTS DATA=_ALL_
  OUT=work.Scratch /*(KEEP=MemName Nobs)*/ NOPRINT NODETAILS;
RUN;
PROC SORT DATA=Scratch NODUPKEY; BY MEMNAME; RUN;
```