TATA CONSULTANCY SERVICES
Experience certainty.

SAS® Training

Introduction to Macros

Topics covered…

- Macro variables
  - %LET
  - SYMPUT / SYMGET
- Writing macros functions
- PROC FCMP

What are macros?

- The macro facility is a tool for simplifying and automating repetitive tasks
- How macros can help:
  - You can make one small change in your code and have SAS echo it throughout your program
  - You can write one piece of code and use it over and over again
  - You can automate your programs and let SAS decide what to do based on actual data values

What are macros?

- The **macro language** is the syntax you use to create and use macros
- A **macro variable** is an efficient way of replacing text strings in SAS code
  - *&name*
- A **macro** is a predefined routine you can use in a SAS program
  - *%name*

Macro Variables

%LET

- %LET is a macro statement that creates a macro variable and assigns it a value
- Useful for simplifying code and automating reports
- Macro variable can be used anywhere in SAS code, such as:
  - Titles/Footnotes
  - IF/WHERE statements
  - Filenames
  - Etc.

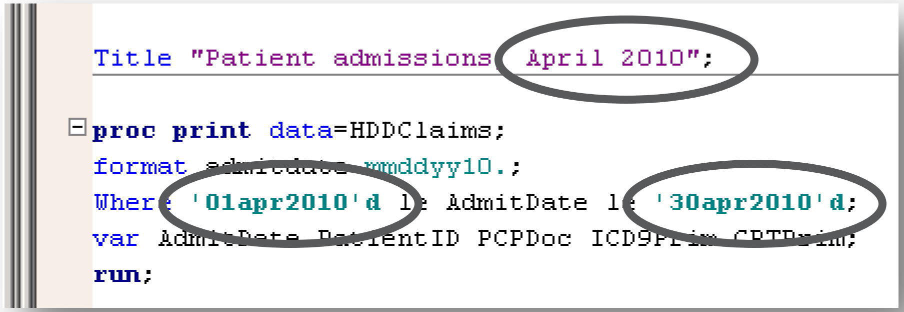
%LET

```
%LET macro-variable = <value>;
```

- ***macro-variable*** : name of new macro variable
- ***value*** : character string or text expression that the macro variable will represent
- Resolve the macro variable later in your code with an ampersand: ***&macro-variable***

%LET

- Report run every month
- Each time you have to change the TITLE statement and the date parameters in the WHERE statement

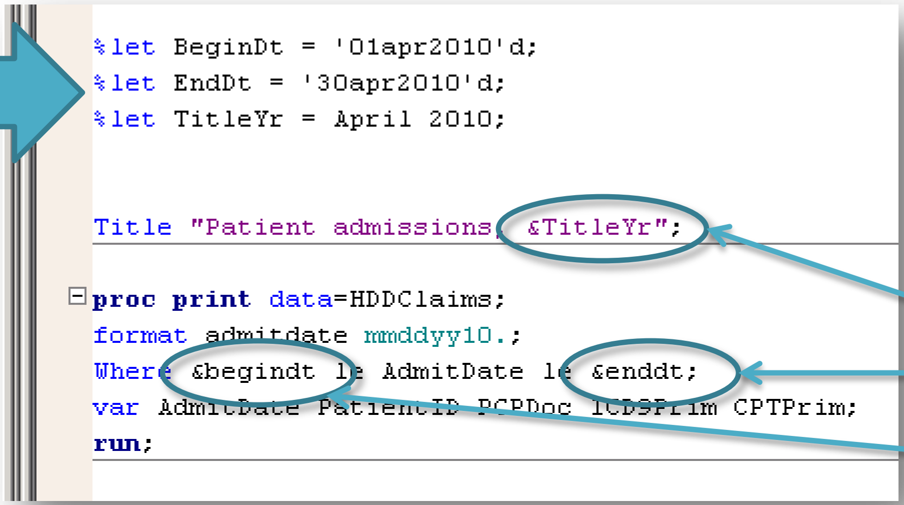


```
Title "Patient admissions April 2010";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where '01apr2010'd le AdmitDate le '30apr2010'd;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;
```

%LET

- Report run every month
- Each time you have to change the TITLE statement and the date parameters in the WHERE statement

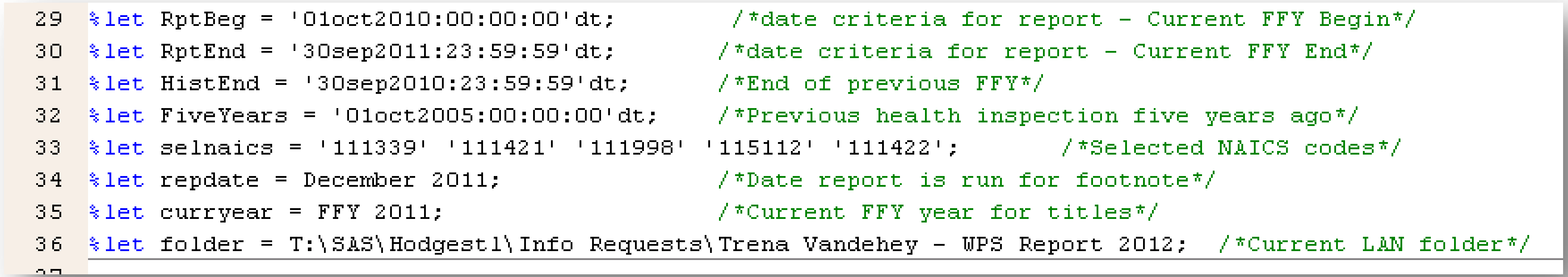
Use %let to define the macro variables



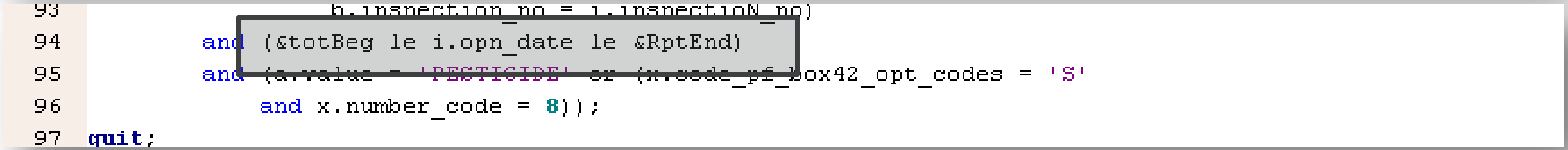
```
%let BeginDt = '01apr2010'd;
%let EndDt = '30apr2010'd;
%let TitleYr = April 2010;


Title "Patient admissions, &TitleYr";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where &begindt le AdmitDate le &enddt;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;
```

Use the macro variables in the program

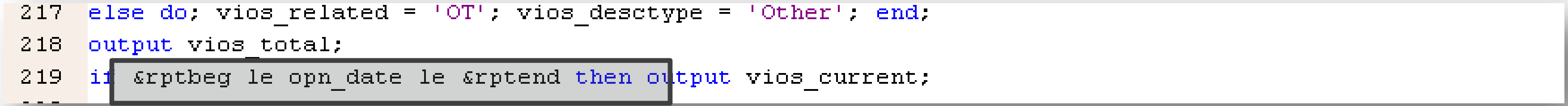
%LET



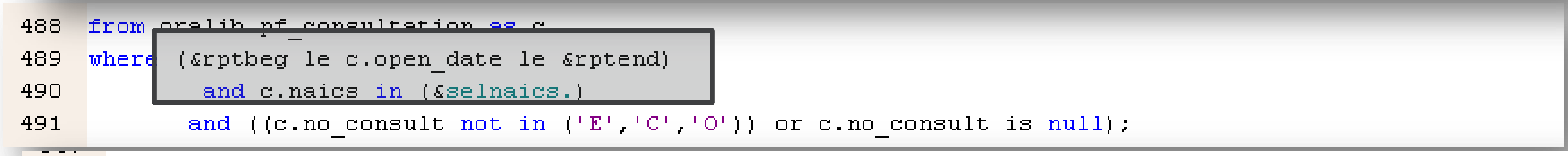
```
%let RptBeg = '01oct2010:00:00:00'dt;        /*date criteria for report - Current FFY Begin*/
%let RptEnd = '30sep2011:23:59:59'dt;       /*date criteria for report - Current FFY End*/
%let HistEnd = '30sep2010:23:59:59'dt;      /*End of previous FFY*/
%let FiveYears = '01oct2005:00:00:00'dt;    /*Previous health inspection five years ago*/
%let selnaics = '111339' '111421' '111998' '115112' '111422';       /*Selected NAICS codes*/
%let repdate = December 2011;               /*Date report is run for footnote*/
%let curryear = FFY 2011;                   /*Current FFY year for titles*/
%let folder = T:\SAS\Hodgest1\Info Requests\Trena Vandehey - WPS Report 2012;  /*Current LAN folder*/
```



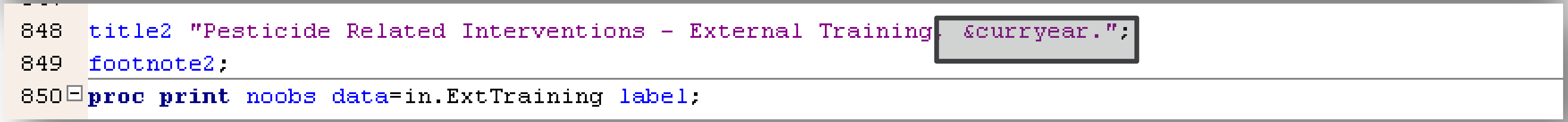
```
                 b.inspection_no = i.inspection_no)
        and (&totBeg le i.opn_date le &RptEnd)
        and (a.value = 'PESTICIDE' or (x.code_pf_box42_opt_codes = 'S'
            and x.number_code = 8));
quit;
```



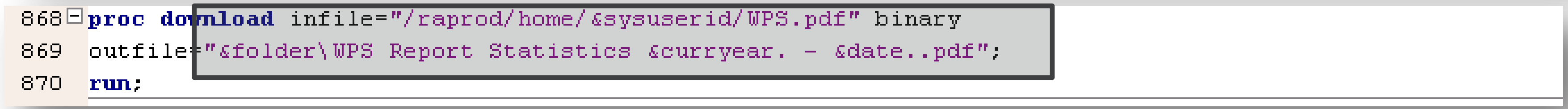
```
else do; vios_related = 'OT'; vios_desctype = 'Other'; end;
output vios_total;
if &rptbeg le opn_date le &rptend then output vios_current;
```



```
from oralib.pf_consultation as c
where (&rptbeg le c.open_date le &rptend)
        and c.naics in (&selnaics.)
        and ((c.no_consult not in ('E','C','O')) or c.no_consult is null);
```



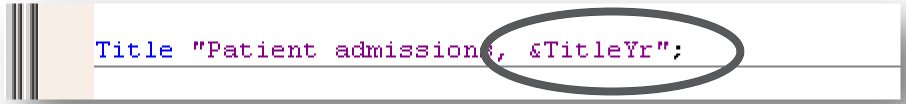
```
title2 "Pesticide Related Interventions - External Training, &curryear.";
footnote2;
proc print noobs data=in.ExtTraining label;
```



```
proc download infile="/raprod/home/&sysuserid/WPS.pdf" binary
outfile="&folder\WPS Report Statistics &curryear. - &date..pdf";
run;
```

Referencing macro variables

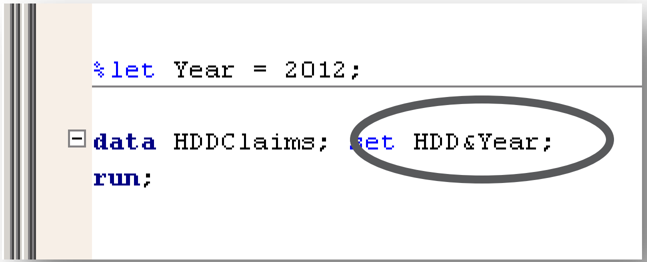
- Reference a macro variable with an ampersand preceding its name
- If within a literal string (such as a title statement), enclose the string in double quotation marks
  - Macro variable references in single quotation marks will not resolve



```
Title "Patient admissions, &TitleYr";
```

Referencing macro variables

- Macro variable references can be placed next to leading or trailing text

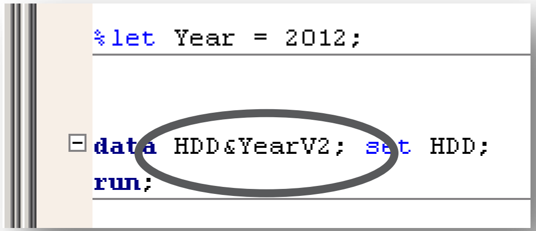


```
%let Year = 2012;

data HDDClaims; set HDD&Year;
run;
```

- Dataset name **HDD&Year** will resolve to **HDD2012**

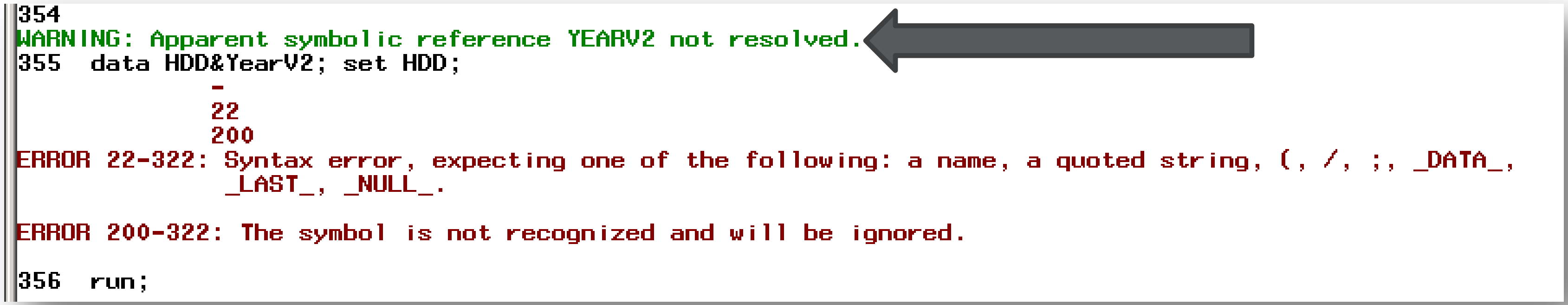
Referencing macro variables

- A macro variable reference with trailing text may not resolve properly



```
%let Year = 2012;


data HDD&YearV2; set HDD;
run;
```

- Meant to create a dataset **HDD2012V2** but got an error instead

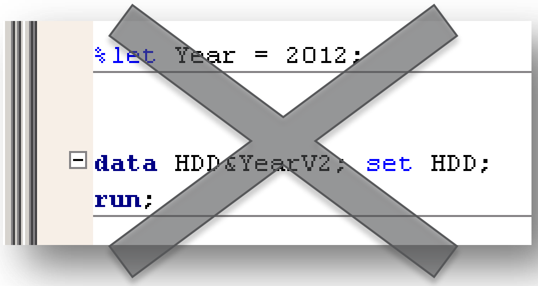


```
354
WARNING: Apparent symbolic reference YEARV2 not resolved.
355  data HDD&YearV2; set HDD;
                 -
                 22
                 200
ERROR 22-322: Syntax error, expecting one of the following: a name, a quoted string, (, /, ;, _DATA_,
              _LAST_, _NULL_.

ERROR 200-322: The symbol is not recognized and will be ignored.

356  run;
```

TATA CONSULTANCY SERVICES
Experience certainty.

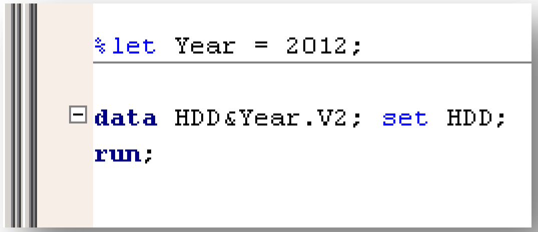
Referencing macro variables

- A macro variable reference with trailing text may not resolve properly



```
%let Year = 2012;

data HDD&YearV2; set HDD;
run;
```

- Use a period as a delimiter to note the end of the macro variable reference

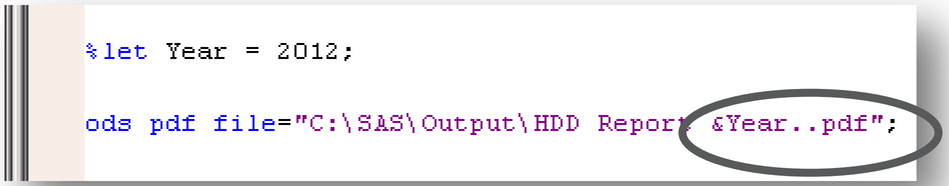


```
%let Year = 2012;

data HDD&Year.V2; set HDD;
run;
```

TATA CONSULTANCY SERVICES
Experience certainty.

Referencing macro variables

- The period delimiter will resolve with the macro variable reference
- You may need a **second** period as part of the original text



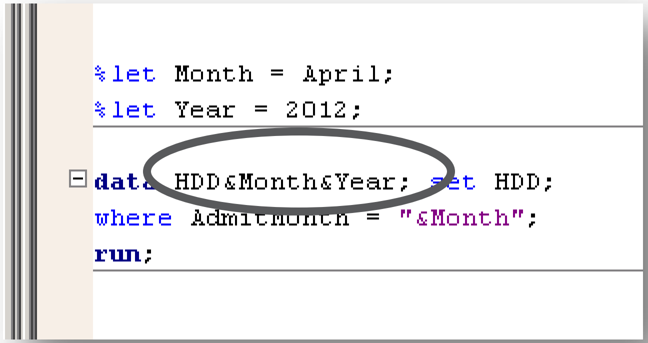
```
%let Year = 2012;

ods pdf file="C:\SAS\Output\HDD Report &Year..pdf";
```

- **HDD Report &Year..pdf** resolves to **HDD Report 2012.pdf**
  - The first period is the delimiter for the macro reference, and the second is part of the text

TATA CONSULTANCY SERVICES
Experience certainty.

Referencing macro variables

- Macro variable references can also be placed next to each other

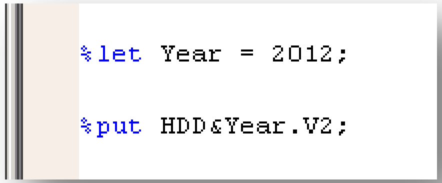


```
%let Month = April;
%let Year = 2012;

data HDD&Month&Year; set HDD;
where AdmitMonth = "&Month";
run;
```

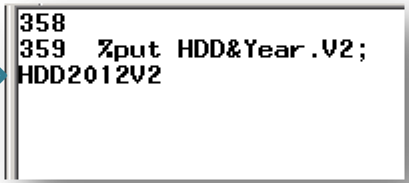
- Dataset name **HDD&Month&Year** will resolve to **HDDApril2012**

Resolving macro variables

- Use **%put** to see the resolved macro reference in your log
  - Can be useful for quickly de-bugging macro references



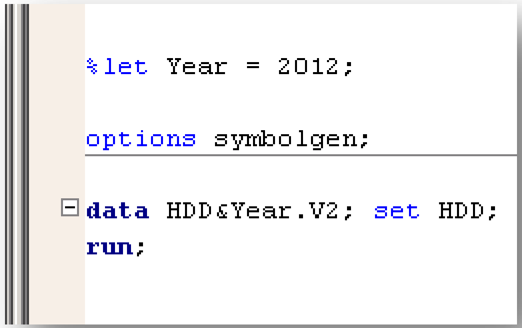
```
%let Year = 2012;

%put HDD&Year.V2;
```



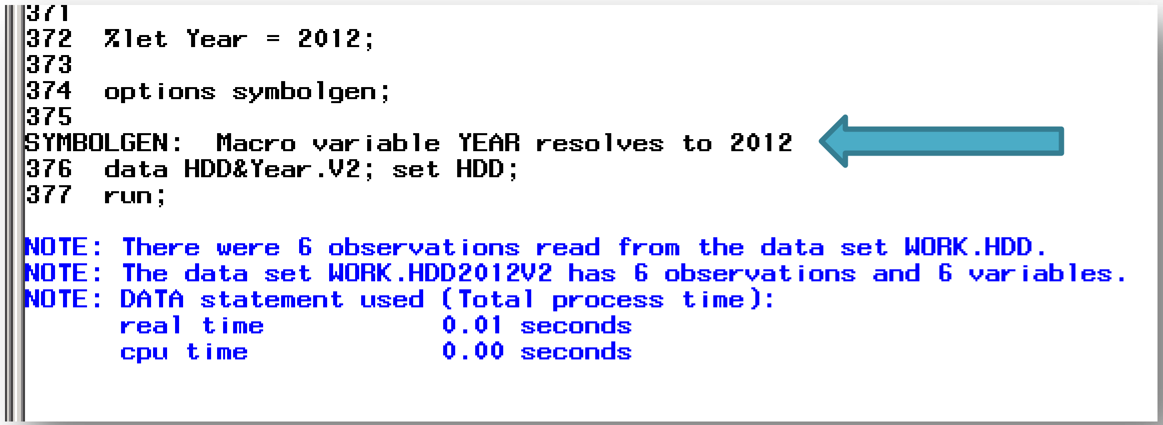
```
358
359   %put HDD&Year.V2;
HDD2012V2
```

Resolving macro variables

- Use the **`symbolgen`** option to display the resolution of macro variable references in the log at the time they are executed
  - Can also be useful for de-bugging



```
%let Year = 2012;

options symbolgen;

data HDD&Year.V2; set HDD;
run;
```

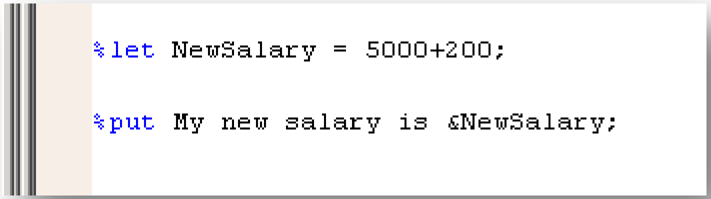


```
371
372   %let Year = 2012;
373
374   options symbolgen;
375
SYMBOLGEN:  Macro variable YEAR resolves to 2012
376   data HDD&Year.V2; set HDD;
377   run;

NOTE: There were 6 observations read from the data set WORK.HDD.
NOTE: The data set WORK.HDD2012V2 has 6 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

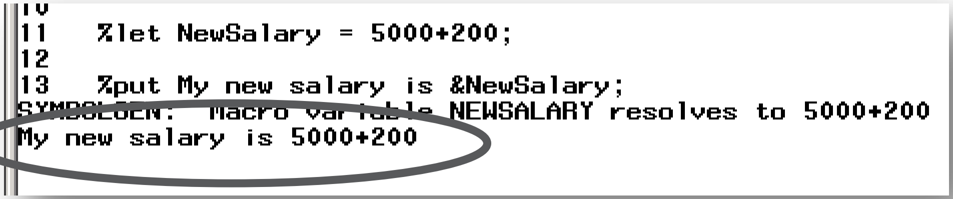
TATA CONSULTANCY SERVICES
Experience certainty.

%EVAL and %SYSEVALF

- Macro variables are **constant text strings**
- Even if it looks like an equation, it will be treated like text



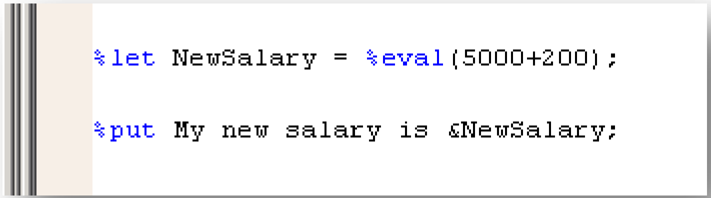
```
%let NewSalary = 5000+200;

%put My new salary is &NewSalary;
```



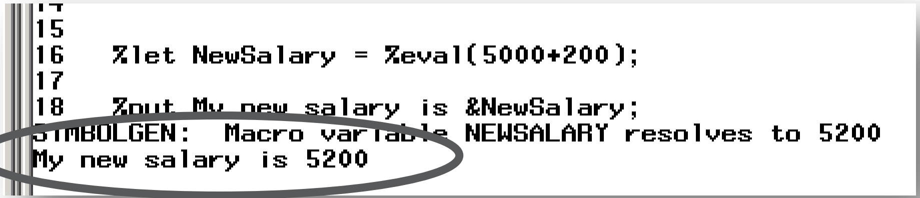
```
   %let NewSalary = 5000+200;

   %put My new salary is &NewSalary;
SYMBOLGEN:  Macro variable NEWSALARY resolves to 5000+200
My new salary is 5000+200
```

%EVAL and %SYSEVALF

- Use the **%eval** function to evaluate the expression using *integer arithmetic*



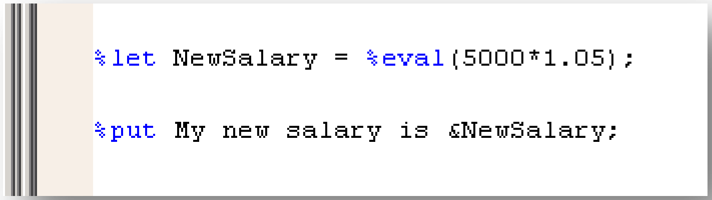
```
%let NewSalary = %eval(5000+200);

%put My new salary is &NewSalary;
```



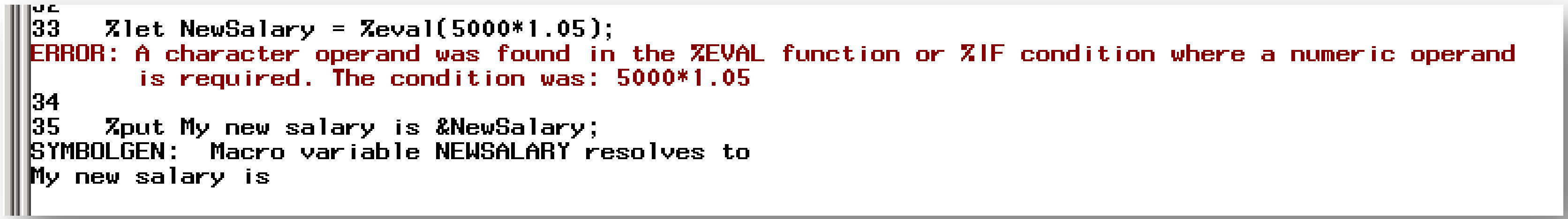
```
15
16    %let NewSalary = %eval(5000+200);
17
18    %put My new salary is &NewSalary;
SYMBOLGEN:  Macro variable NEWSALARY resolves to 5200
My new salary is 5200
```

%EVAL and %SYSEVALF

- **`%eval`** cannot be used with floating-point numbers (numbers with a decimal point)



```
%let NewSalary = %eval(5000*1.05);

%put My new salary is &NewSalary;
```

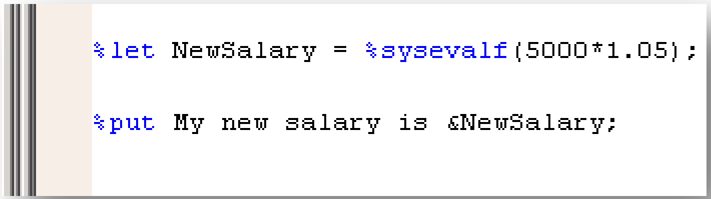


```
33   %let NewSalary = %eval(5000*1.05);
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand
       is required. The condition was: 5000*1.05
34
35   %put My new salary is &NewSalary;
SYMBOLGEN:  Macro variable NEWSALARY resolves to
My new salary is
```

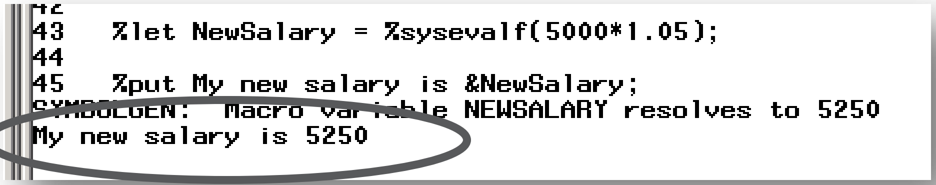
TATA CONSULTANCY SERVICES
Experience certainty.

%EVAL and %SYSEVALF

- Use the **`%sysevalf`** function for floating-point arithmetic



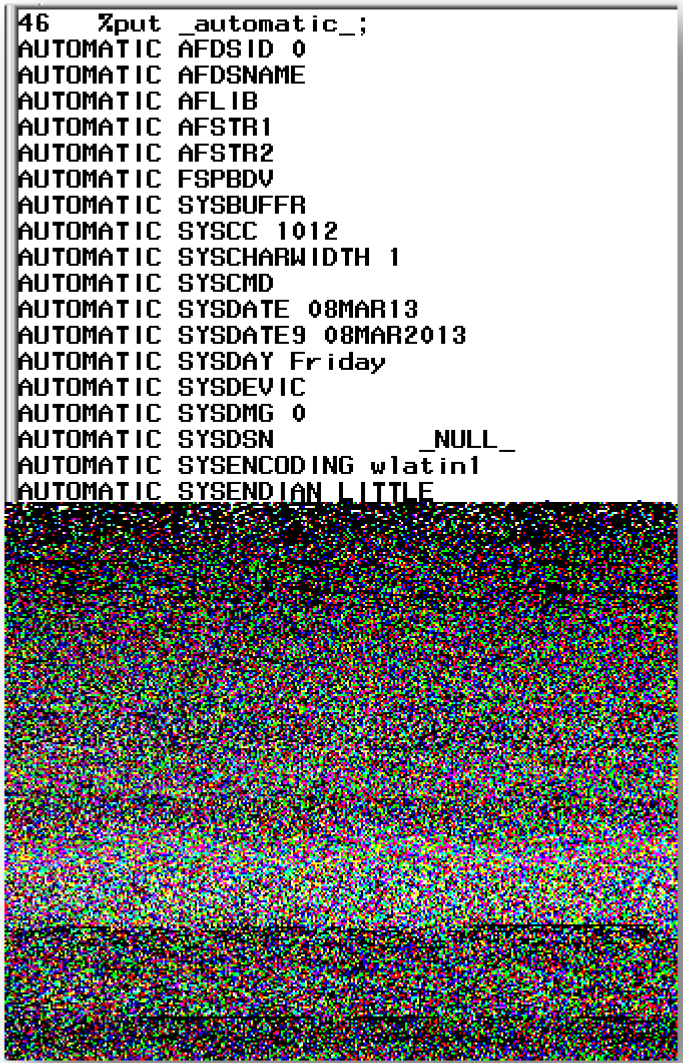
```
%let NewSalary = %sysevalf(5000*1.05);

%put My new salary is &NewSalary;
```



```
   %let NewSalary = %sysevalf(5000*1.05);

   %put My new salary is &NewSalary;
SYMBOLGEN:  Macro variable NEWSALARY resolves to 5250
My new salary is 5250
```

Automatic macro variables

- SAS has built in macro variables called automatic macro variables
  - They are created at the beginning of every SAS session
  - Use **`%put _automatic_;`** to see the list in the log

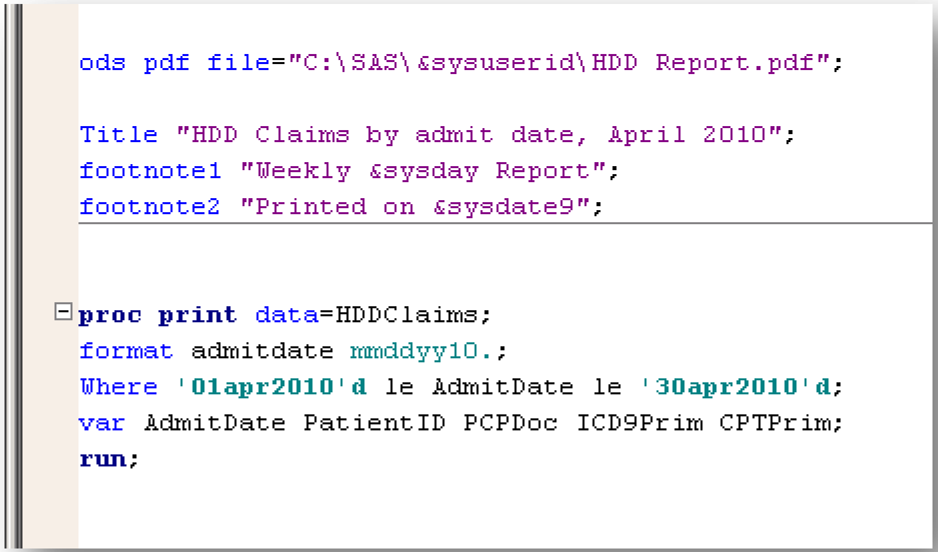


```
46    %put _automatic_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 1012
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 08MAR13
AUTOMATIC SYSDATE9 08MAR2013
AUTOMATIC SYSDAY Friday
AUTOMATIC SYSDEVIC
AUTOMATIC SYSDMG 0
AUTOMATIC SYSDSN          _NULL_
AUTOMATIC SYSENCODING wlatin1
AUTOMATIC SYSENDIAN LITTLE
```

TATA CONSULTANCY SERVICES
Experience certainty.

Automatic macro variables

- Some potentially useful automatic macro variables:
  - SYSDATE
  - SYSDATE9
  - SYSDAY
  - SYSTIME
  - SYSUSERID



```
ods pdf file="C:\SAS\&sysuserid\HDD Report.pdf";

Title "HDD Claims by admit date, April 2010";
footnote1 "Weekly &sysday Report";
footnote2 "Printed on &sysdate9";


proc print data=HDDClaims;
format admitdate mmddyy10.;
Where '01apr2010'd le AdmitDate le '30apr2010'd;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;
```

TATA CONSULTANCY SERVICES
Experience certainty.

Symput/Symget

CALL SYMPUTX

- How do you turn today's date into a macro variable?
- ***Option 1:*** Use the SYSDATE or SYSDATE9 automatic variables
  - Problem – SYSDATE and SYSDATE9 are not dates, but text strings. What if you don't like that format?
- ***Option 2:*** Use CALL SYMPUTX

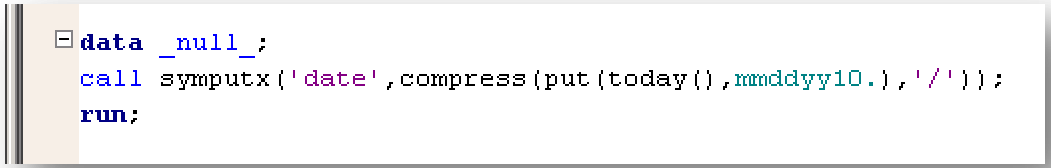
CALL SYMPUTX

- CALL SYMPUT and CALL SYMPUTX assign a value to a macro variable (similar to %LET)

- Unlike %LET, the value can be based on a calculation, algorithm or dataset variable

CALL SYMPUTX

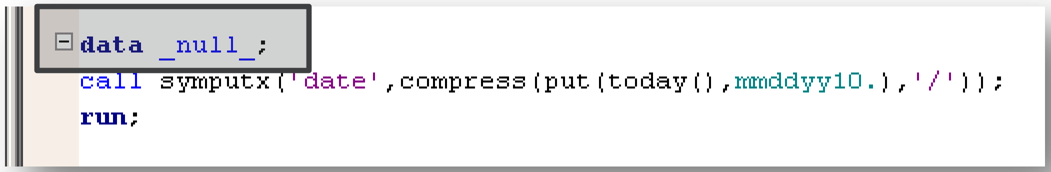
- Below is an example of CALL SYMPUTX
- The end result is macro variable that contains today’s date with no special characters
  - Useful for adding a date to filenames



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

CALL SYMPUTX

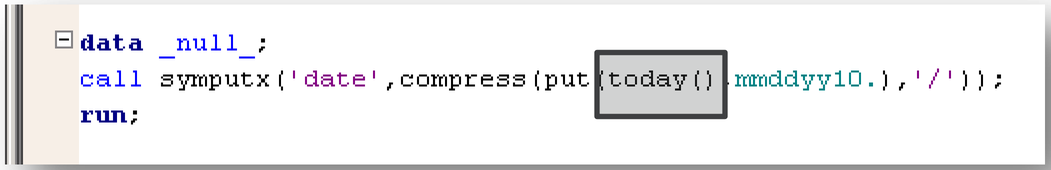
- **DATA _NULL_**
  - Allows you to execute a DATA step without creating a new dataset



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

CALL SYMPUTX

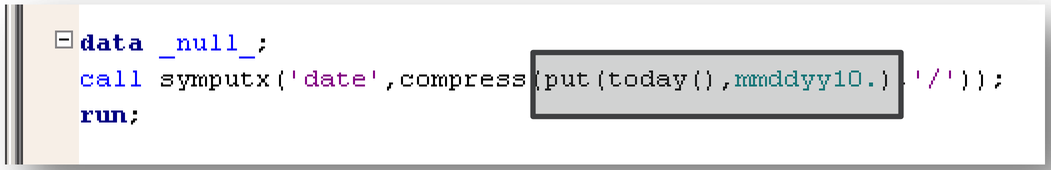
- **TODAY()**
  - Computes the current date
  - Example: March 26, 2013



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

CALL SYMPUTX

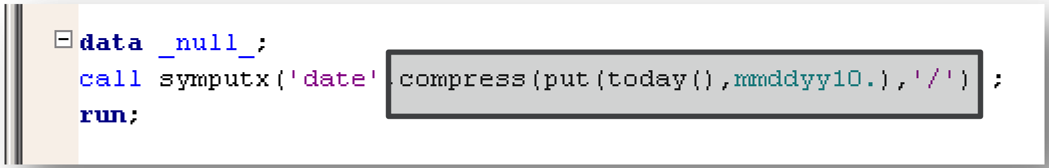
- **PUT (**today(), MMDDYY10.)
  - Converts today's date into a character string
  - The string will have the appearance of the MMDDYY10. date format
  - Example: 03/26/2013



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

CALL SYMPUTX

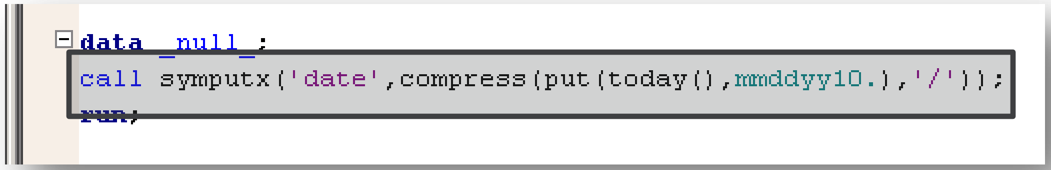
- **COMPRESS(**put(today(), mmddyy10.)**, '/')**
  - Removes the forward slashes from the text string
  - Example: 03262013



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

CALL SYMPUTX

- **CALL SYMPUTX('DATE',** compress(put(today(), mmddyy10.), '/')**)**
  - Assigns this value to a macro variable called DATE
  - Similar to **%LET date = 03262013;**



```
data _null_;
call symputx('date',compress(put(today(),mmddyy10.),'/'));
run;
```

SYMGET function

- Conversely, the SYMGET function returns the value of a macro variable during DATA step execution

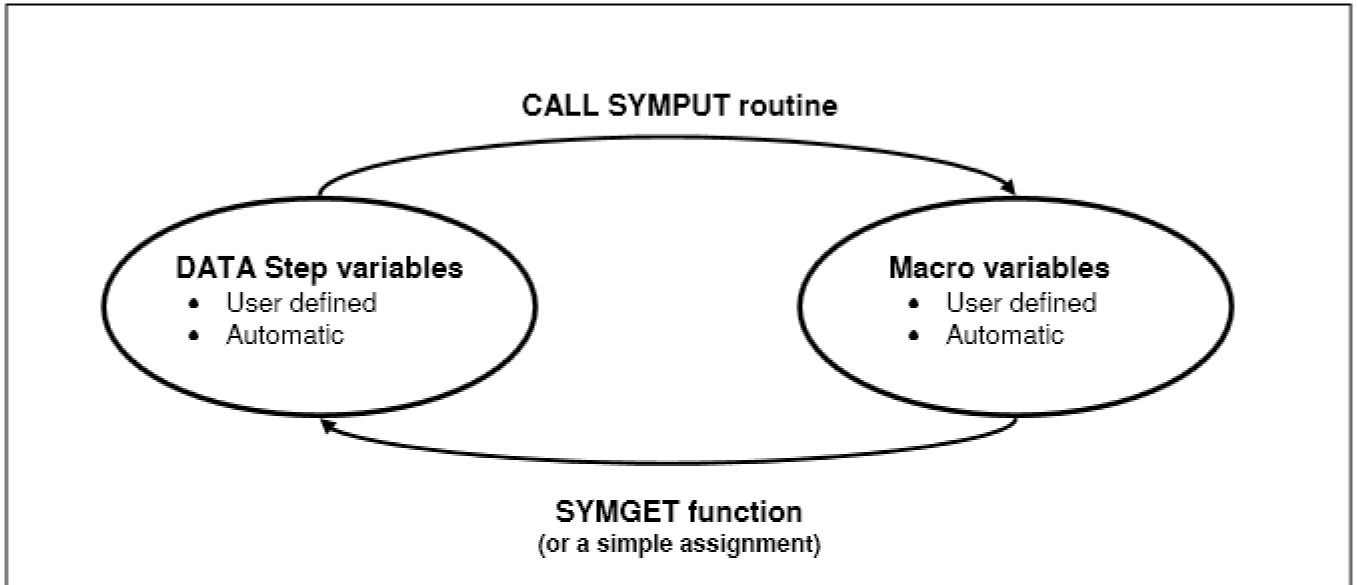
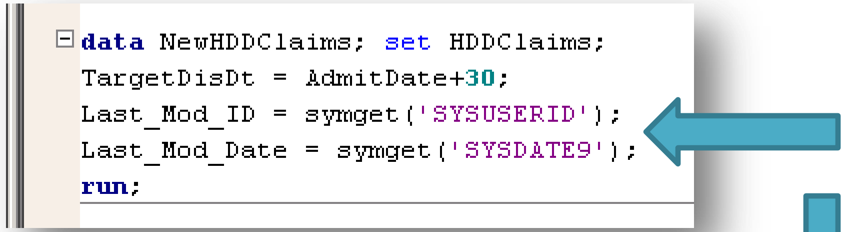


Figure 1. Schematic representation of communication between the DATA step and the macro facility

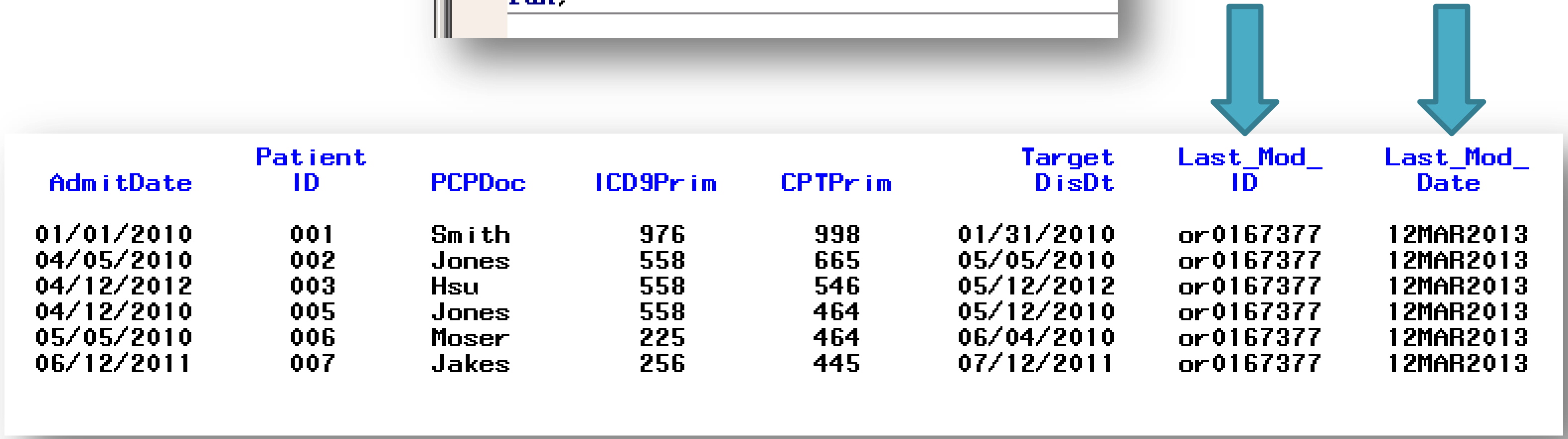
Graphic from "SYMPUT and SYMGET: Getting DATA Step Variables and Macro Variables to Share" by Christianna Willaims

SYMGET function

- Example: Have the DATA step store the SYSUSERID and SYSDATE to keep record of who last modified the dataset



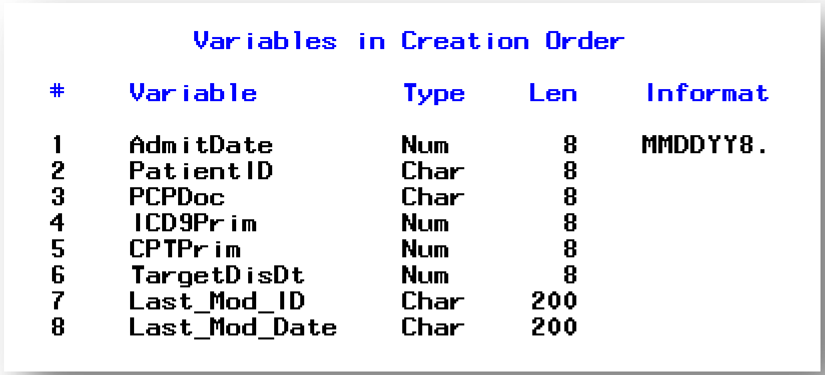
```
data NewHDDClaims; set HDDClaims;
TargetDisDt = AdmitDate+30;
Last_Mod_ID = symget('SYSUSERID');
Last_Mod_Date = symget('SYSDATE9');
run;
```



AdmitDate	Patient ID	PCPDoc	ICD9Prim	CPTPrim	Target DisDt	Last_Mod_ ID	Last_Mod_ Date
01/01/2010	001	Smith	976	998	01/31/2010	or0167377	12MAR2013
04/05/2010	002	Jones	558	665	05/05/2010	or0167377	12MAR2013
04/12/2012	003	Hsu	558	546	05/12/2012	or0167377	12MAR2013
04/12/2010	005	Jones	558	464	05/12/2010	or0167377	12MAR2013
05/05/2010	006	Moser	225	464	06/04/2010	or0167377	12MAR2013
06/12/2011	007	Jakes	256	445	07/12/2011	or0167377	12MAR2013

SYMGET function

- The SYMGET function, by default, stores the new variables as **character** variables with a length of **200**



```
                Variables in Creation Order

#     Variable            Type     Len     Informat

1     AdmitDate           Num        8     MMDDYY8.
2     PatientID           Char       8
3     PCPDoc              Char       8
4     ICD9Prim            Num        8
5     CPTPrim             Num        8
6     TargetDisDt         Num        8
7     Last_Mod_ID         Char     200
8     Last_Mod_Date       Char     200
```

Macros

Macro programs

- Anytime you find yourself repeating tasks or programs, you might want to consider creating a macro
- Macros are simply a group of SAS statements with a name
- Instead of re-typing the statements, you use the macro name to invoke the code
- The invoked macro will write the code to your program
  - Think of it as an advanced version of find/replace

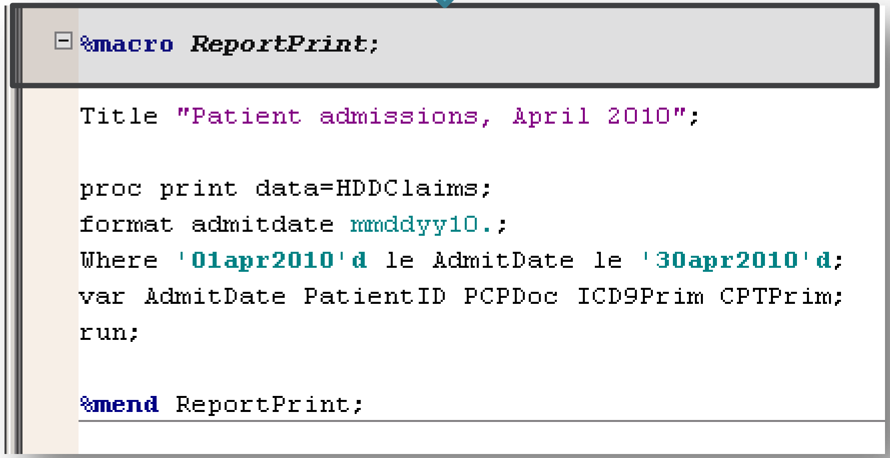
Macro programs

```
%MACRO macro-name <(parameters)>;
        macro-text
%MEND macro-name;
```

- ***macro-name*** : name of new macro program
- ***parameters*** : local macro variables, whose values you specify when you invoke the macro
- Resolve the macro variable later in your code with a percent sign: ***%macro-name <(parameters)>***

Writing a macro

%MACRO
Begins the macro definition.
Includes macro name and (if applicable) parameters

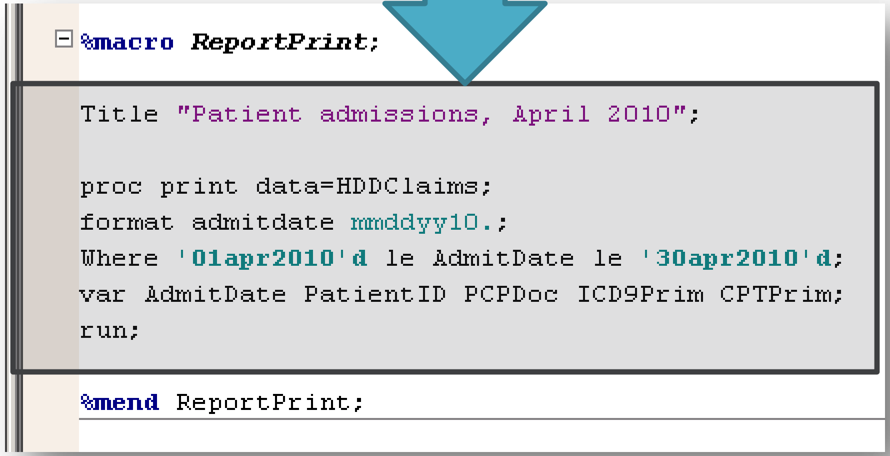


```
%macro ReportPrint;

Title "Patient admissions, April 2010";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where '01apr2010'd le AdmitDate le '30apr2010'd;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;

%mend ReportPrint;
```

TATA CONSULTANCY SERVICES
Experience certainty.

Writing a macro

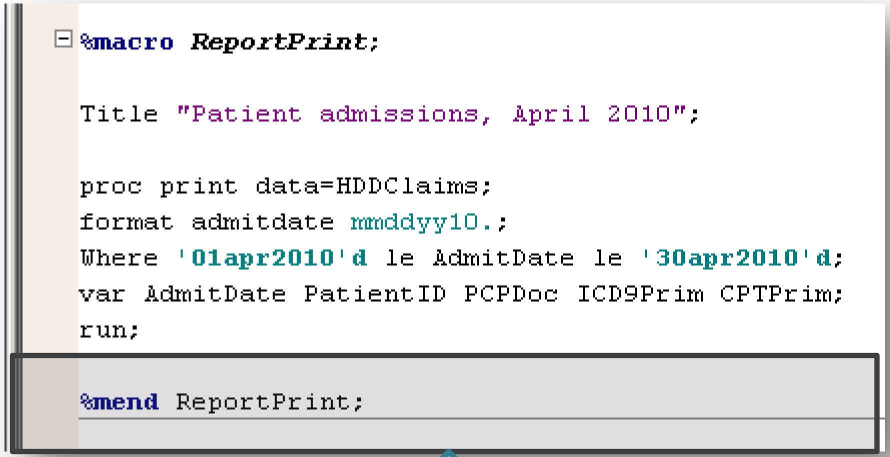
Macro Text
Any and all statements to include as part of the macro



```
%macro ReportPrint;

Title "Patient admissions, April 2010";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where '01apr2010'd le AdmitDate le '30apr2010'd;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;

%mend ReportPrint;
```

TATA CONSULTANCY SERVICES
Experience certainty.

Writing a macro

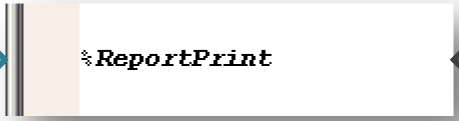


```
%macro ReportPrint;

Title "Patient admissions, April 2010";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where '01apr2010'd le AdmitDate le '30apr2010'd;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;

%mend ReportPrint;
```

%MEND
Ends the macro definition

Invoking a macro

Invoke the macro anywhere in open code when you want to use it.

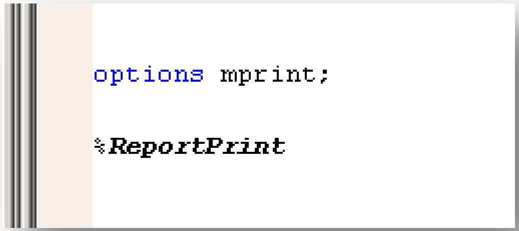


```
%ReportPrint
```

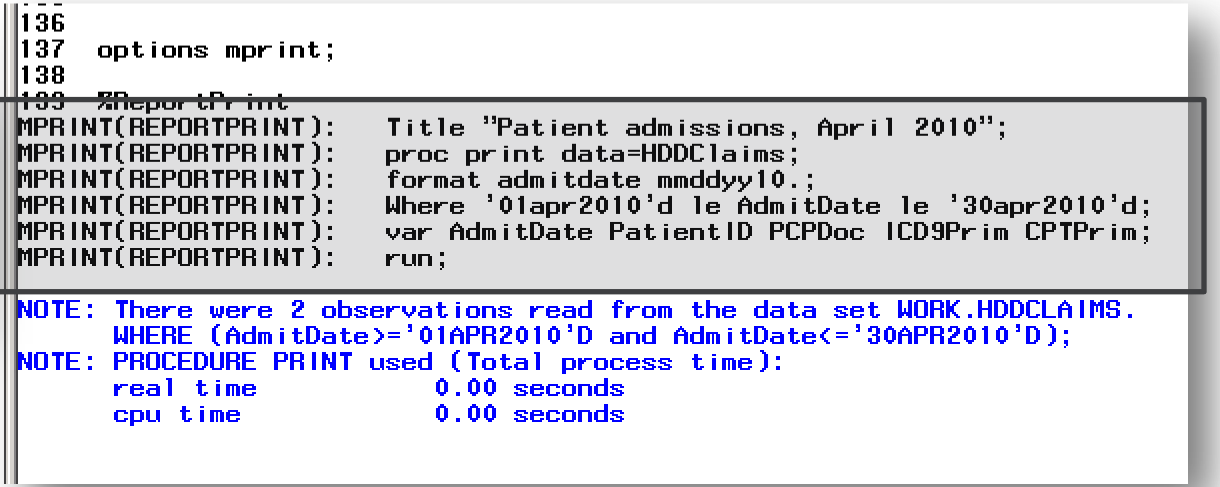
Do ***NOT*** put a semicolon after a macro invocation. (This may interfere with the macro code)

MPRINT option

- Use the **`mprint`** option to display the resolution of the macro in the log at the time they are executed
  - Without it your log will only show the macro call



```
options mprint;

%ReportPrint
```

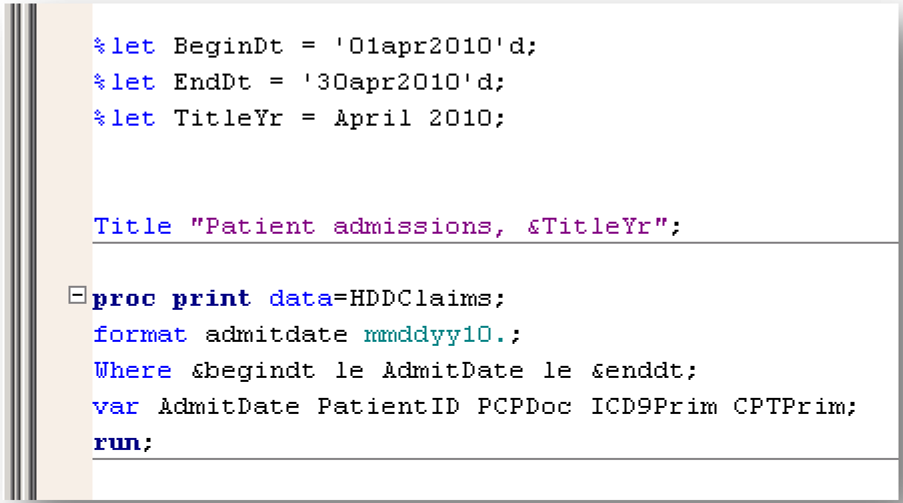


```
136
137   options mprint;
138
139   %ReportPrint
MPRINT(REPORTPRINT):   Title "Patient admissions, April 2010";
MPRINT(REPORTPRINT):   proc print data=HDDClaims;
MPRINT(REPORTPRINT):   format admitdate mmddyy10.;
MPRINT(REPORTPRINT):   Where '01apr2010'd le AdmitDate le '30apr2010'd;
MPRINT(REPORTPRINT):   var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
MPRINT(REPORTPRINT):   run;

NOTE: There were 2 observations read from the data set WORK.HDDCLAIMS.
      WHERE (AdmitDate>='01APR2010'D and AdmitDate<='30APR2010'D);
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

TATA CONSULTANCY SERVICES
Experience certainty.

Parameters

- Use parameters to increase the flexibility of your macro
- Similar to %LET

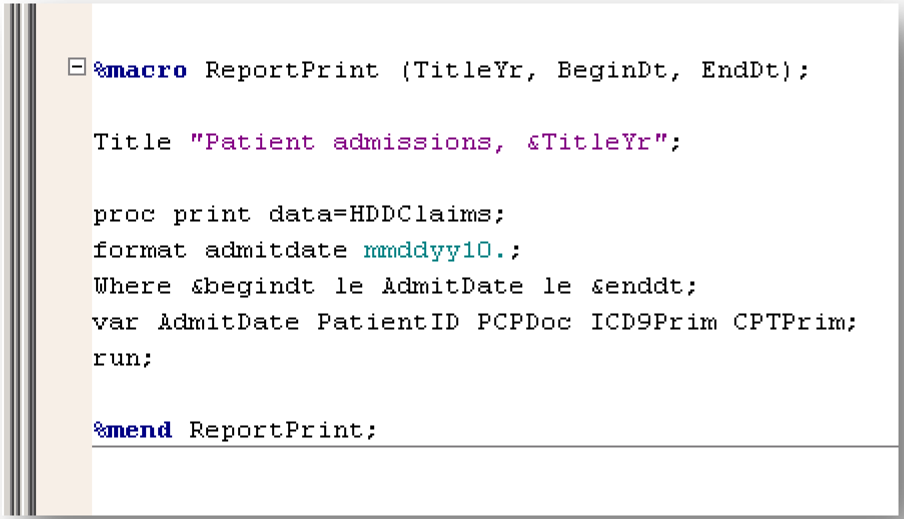


```
%let BeginDt = '01apr2010'd;
%let EndDt = '30apr2010'd;
%let TitleYr = April 2010;


Title "Patient admissions, &TitleYr";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where &begindt le AdmitDate le &enddt;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;
```

TATA CONSULTANCY SERVICES
Experience certainty.

Parameters

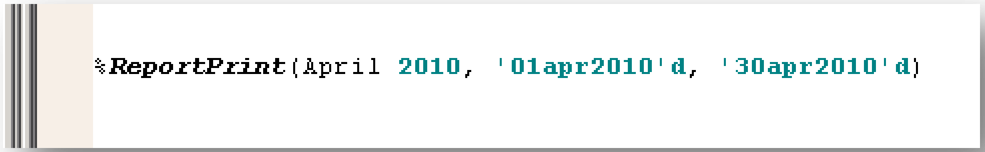
- Use parameters to increase the flexibility of your macro
- Similar to %LET



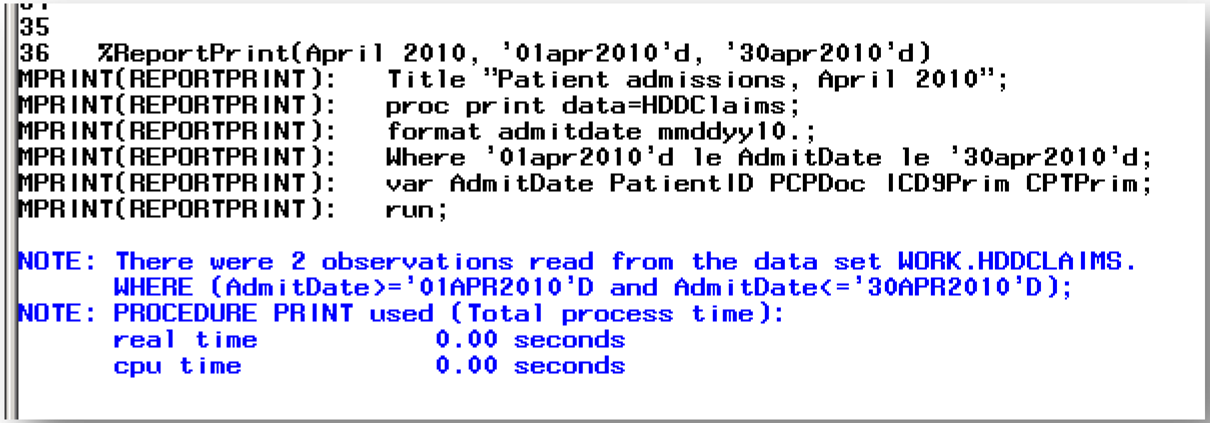
```
%macro ReportPrint (TitleYr, BeginDt, EndDt);

Title "Patient admissions, &TitleYr";

proc print data=HDDClaims;
format admitdate mmddyy10.;
Where &begindt le AdmitDate le &enddt;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;

%mend ReportPrint;
```

Parameters

- Use parameters to increase the flexibility of your macro
- Similar to %LET



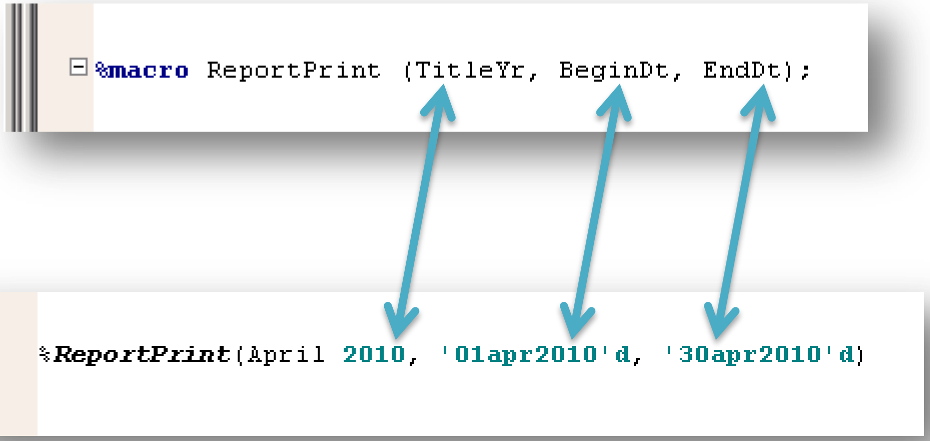
```
%ReportPrint(April 2010, '01apr2010'd, '30apr2010'd)
```



```
35
36   %ReportPrint(April 2010, '01apr2010'd, '30apr2010'd)
MPRINT(REPORTPRINT):   Title "Patient admissions, April 2010";
MPRINT(REPORTPRINT):   proc print data=HDDClaims;
MPRINT(REPORTPRINT):   format admitdate mmddyy10.;
MPRINT(REPORTPRINT):   Where '01apr2010'd le AdmitDate le '30apr2010'd;
MPRINT(REPORTPRINT):   var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
MPRINT(REPORTPRINT):   run;

NOTE: There were 2 observations read from the data set WORK.HDDCLAIMS.
      WHERE (AdmitDate>='01APR2010'D and AdmitDate<='30APR2010'D);
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Parameters

- Positional parameters
  - The order of the values specified in macro invocation matches the order listed in the %MACRO statement

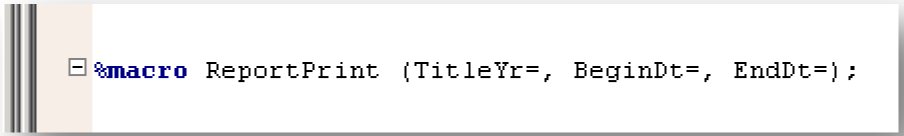


```
%macro ReportPrint (TitleYr, BeginDt, EndDt);
```

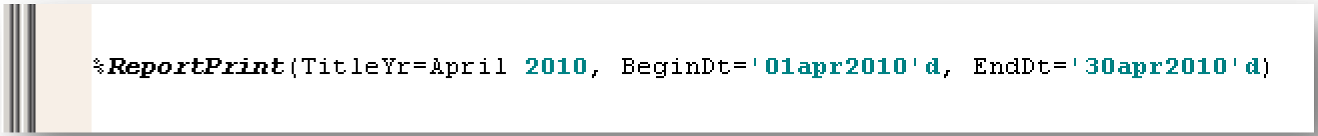
```
%ReportPrint(April 2010, '01apr2010'd, '30apr2010'd)
```

Parameters

- Keyword parameters
  - The values are specified in macro invocation using the macro name followed by an equal sign



```
%macro ReportPrint (TitleYr=, BeginDt=, EndDt=);
```

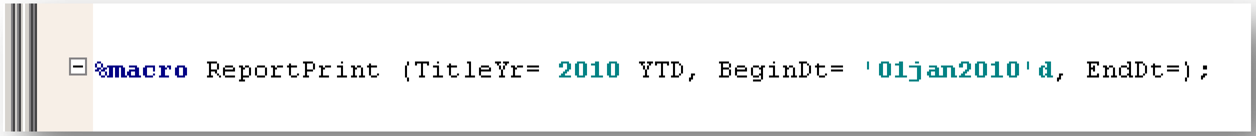


```
%ReportPrint(TitleYr=April 2010, BeginDt='01apr2010'd, EndDt='30apr2010'd)
```

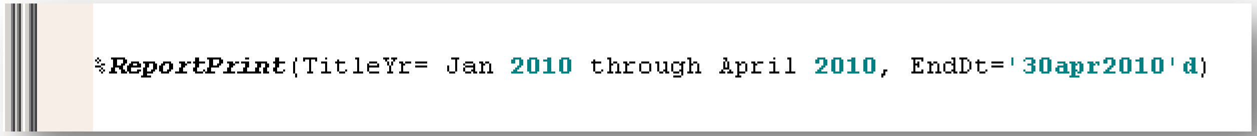
TATA CONSULTANCY SERVICES
Experience certainty.

Parameters

- Keyword parameters
  - Default values can be specified in the %MACRO statement for some or all of the macro variables
  - Default values can be overridden during macro invocation



```
%macro ReportPrint (TitleYr= 2010 YTD, BeginDt= '01jan2010'd, EndDt=);
```



```
%ReportPrint(TitleYr= Jan 2010 through April 2010, EndDt='30apr2010'd)
```

Global vs Local

- Macro parameters are **local** macro variables
  - They are defined in macro code
  - They can only be used within the macro that defines them
- Other macro variables are **global** macro variables
  - They are defined in open code (non-macro code)
  - They can be used anywhere in open code and in macro code
  - Examples include automatic macro variables, variables created with %LET, etc.
- **Best practice:** Do not create a local and global macro variable with the same name

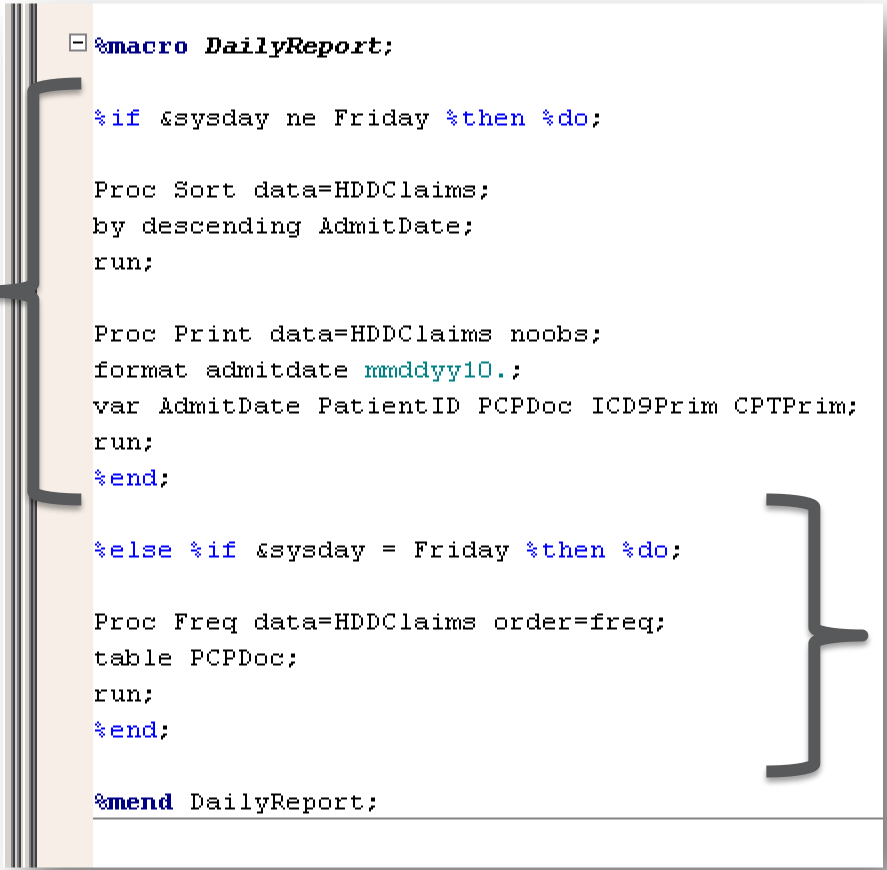
Conditional Logic

%IF, %THEN, %ELSE

- **`%IF`**, **`%THEN`**, **`%ELSE`** is similar to **`IF`**, **`THEN`**, **`ELSE`**, but they are not the exactly same
  - **`IF`**, **`THEN`**, **`ELSE`** conditionally executes SAS statements in the DATA step
  - **`%IF`**, **`%THEN`**, **`%ELSE`** conditionally generates text in a macro

%IF, %THEN, %ELSE

If the SYSDAY **is not** Friday, the macro will generate a detailed report with PROC Print



```
%macro DailyReport;

%if &sysday ne Friday %then %do;

Proc Sort data=HDDClaims;
by descending AdmitDate;
run;

Proc Print data=HDDClaims noobs;
format admitdate mmddyy10.;
var AdmitDate PatientID PCPDoc ICD9Prim CPTPrim;
run;
%end;

%else %if &sysday = Friday %then %do;

Proc Freq data=HDDClaims order=freq;
table PCPDoc;
run;
%end;

%mend DailyReport;
```

If the SYSDAY **is** Friday, the macro will generate a summary report with PROC Freq

PROC FCMP

PROC FCMP

- PROC FCMP stands for SAS Function Compiler Procedure
- Create and reuse functions, CALL routines, and subroutines
- Only available in SAS version 9 and above

PROC FCMP

- Some benefits of PROC FCMP over Macros:
  - Macros are not independent from the main program
  - Macros can get tripped up with global vs. local variables
  - PROC FCMP written in (essentially) DATA step syntax
  - Substantial use of macros can result in illegible code

Example of a function

```
ROUND(argument, <rounding-unit>)
```

- Rounds a number to the selected rounding unit
- ***argument*** : numeric value to be rounded
- ***rounding-unit*** : optional increment for rounding
  - Default is 1 (nearest integer)

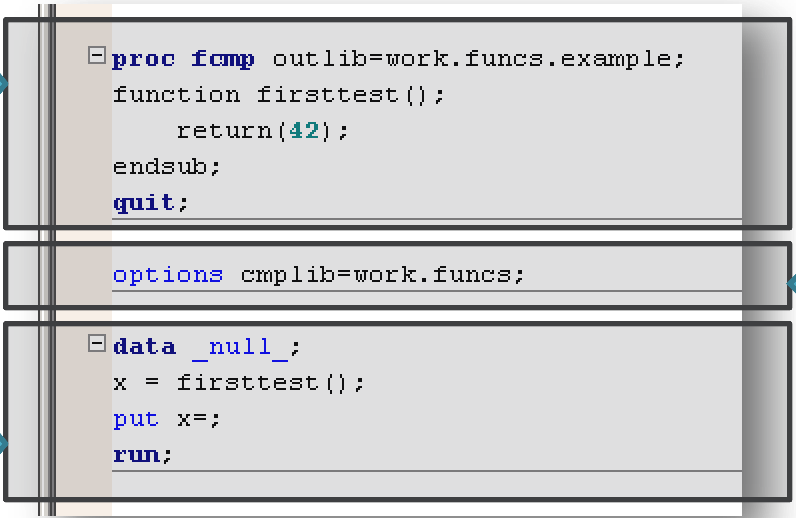
Function	Result
`round(156.826, 1)`	157
`round(156.826, 10)`	160
`round(156.826, .01)`	156.83

PROC FCMP

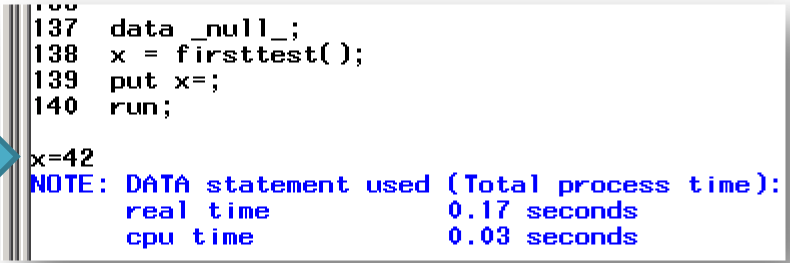
Use `PROC FCMP` to create a new function: `firsttest()`

Use `DATA _NULL_` step to test the new function

Use `cmplib=` option to tell SAS where to find user defined functions



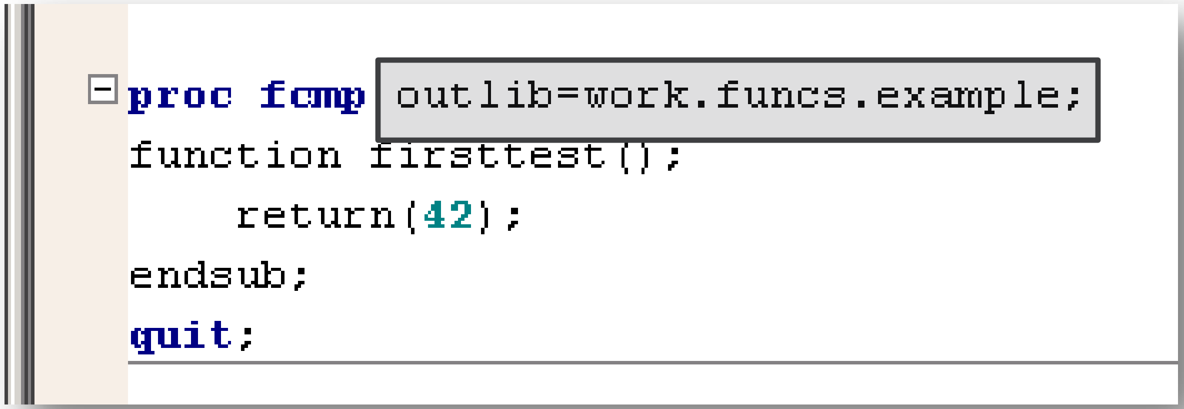
```
proc fcmp outlib=work.funcs.example;
function firsttest();
     return(42);
endsub;
quit;

options cmplib=work.funcs;

data _null_;
x = firsttest();
put x=;
run;
```



```
137   data _null_;
138   x = firsttest( );
139   put x=;
140   run;

x=42
NOTE: DATA statement used (Total process time):
      real time           0.17 seconds
      cpu time            0.03 seconds
```

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP



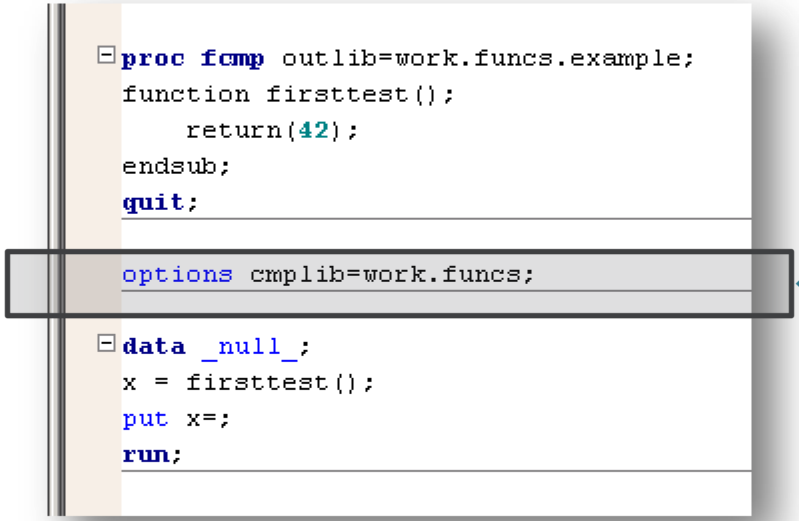
```
proc fcmp outlib=work.funcs.example;
function firsttest();
     return(42);
endsub;
quit;
```

Save functions to a package in an output dataset so they can be used later.
(A package is a collection of routines.)

OUTLIB=*library.dataset.package*

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP



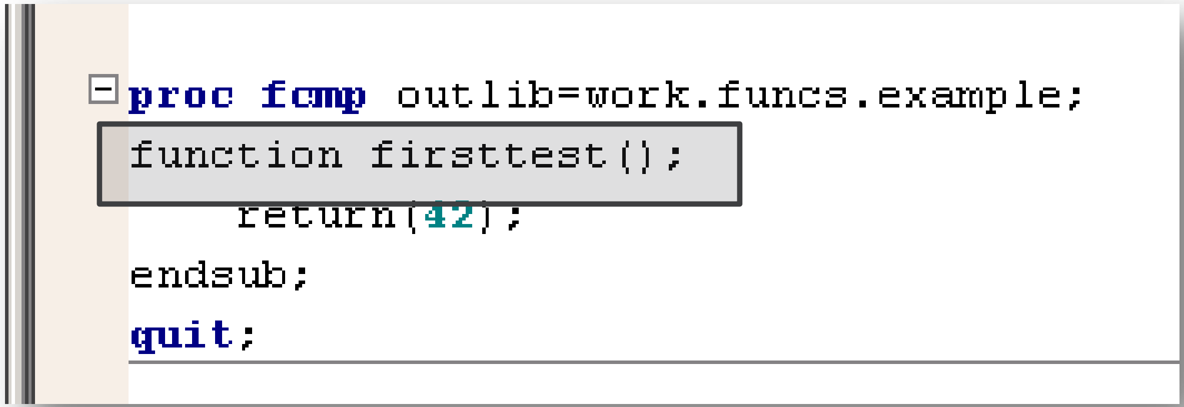
```
proc fcmp outlib=work.funcs.example;
function firsttest();
    return(42);
endsub;
quit;

options cmplib=work.funcs;

data _null_;
x = firsttest();
put x=;
run;
```

Use **`cmplib=`** option to tell SAS where to find user defined functions

Specify where to look for previously compiled functions and subroutines.

`CMPLIB=`*`library.dataset`*

PROC FCMP

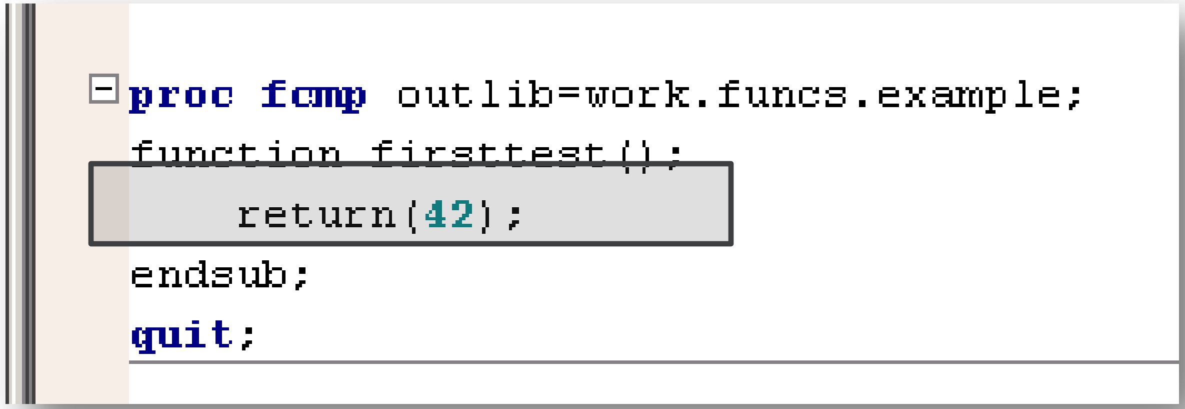


```
proc fcmp outlib=work.funcs.example;
function firsttest();
     return(42);
endsub;
quit;
```

Name your function and specify one or more arguments for the function. (In this example there are no arguments.)

FUNCTION ***function-name* (*argument-1*,… *argument-n*);**

PROC FCMP



```
proc fcmp outlib=work.funcs.example;
function firsttest();
     return(42);
endsub;
quit;
```

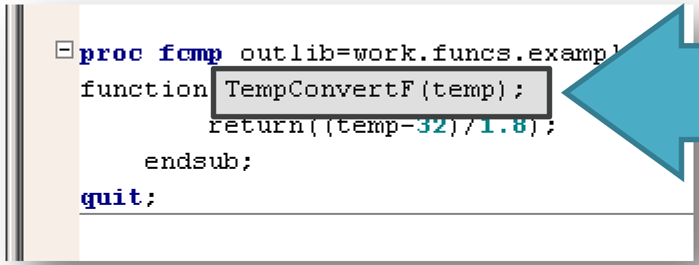
Specify the value that is returned from the function. (In this example, the answer to the ultimate question.)

This can also be an arithmetic expression or the result of data manipulation or conditional logic.

RETURN (***expression***);

PROC FCMP example

- Create a function that converts temperature from Fahrenheit to Celsius



```
proc fcmp outlib=work.funcs.exampl
function TempConvertF(temp);
         return((temp-32)/1.8);
     endsub;
quit;
```

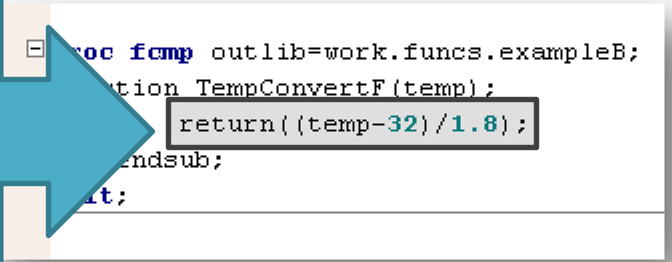
The new function **`TempConvertF`** will have one argument: **`temp`**

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP example

- Create a function that converts temperature from Fahrenheit to Celsius

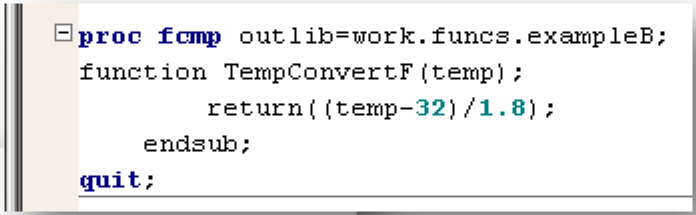
The resulting value will be temperature in Celsius:
`(Temp – 32) / 1.8`



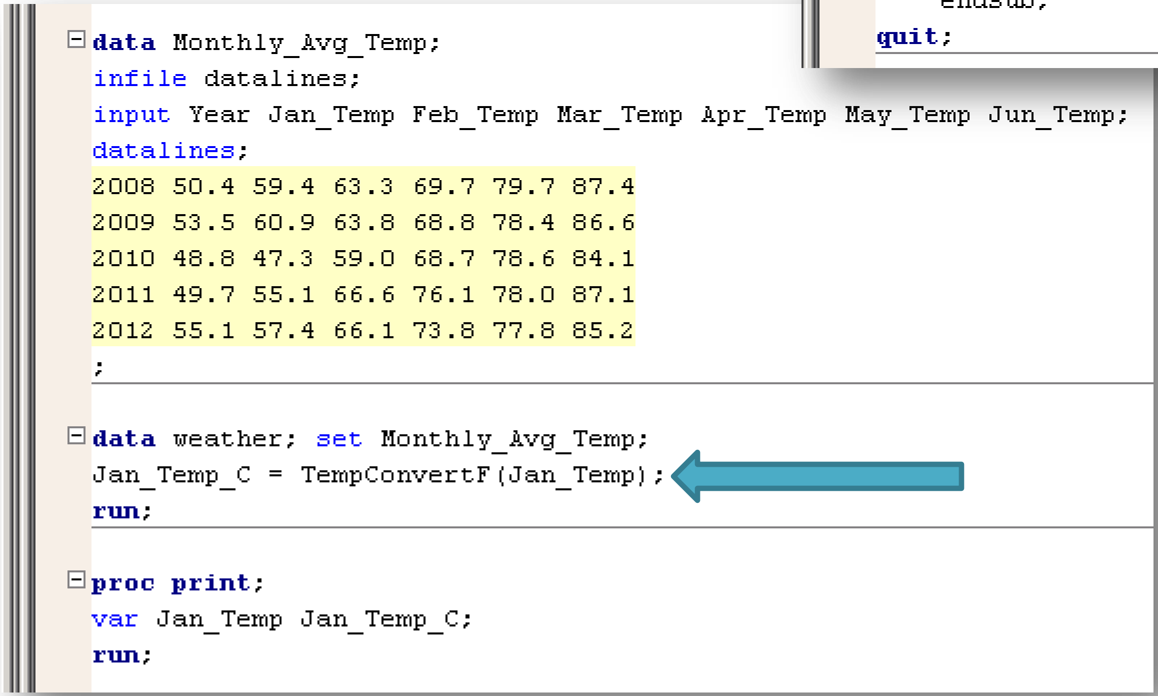
```
roc fcmp outlib=work.funcs.exampleB;
 tion TempConvertF(temp);
        return((temp-32)/1.8);
 ndsub;
it;
```

PROC FCMP example

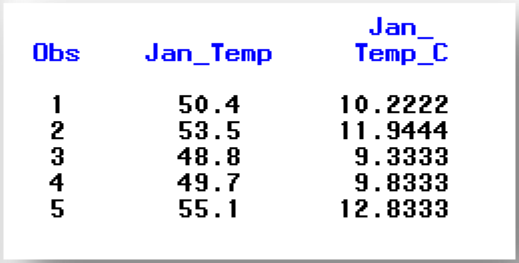
- Create a function that converts temperature from Fahrenheit to Celsius



```
proc fcmp outlib=work.funcs.exampleB;
function TempConvertF(temp);
        return((temp-32)/1.8);
    endsub;
quit;
```



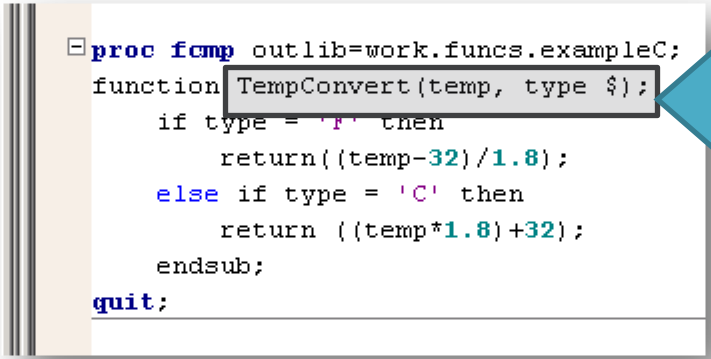
```
data Monthly_Avg_Temp;
infile datalines;
input Year Jan_Temp Feb_Temp Mar_Temp Apr_Temp May_Temp Jun_Temp;
datalines;
2008 50.4 59.4 63.3 69.7 79.7 87.4
2009 53.5 60.9 63.8 68.8 78.4 86.6
2010 48.8 47.3 59.0 68.7 78.6 84.1
2011 49.7 55.1 66.6 76.1 78.0 87.1
2012 55.1 57.4 66.1 73.8 77.8 85.2
;

data weather; set Monthly_Avg_Temp;
Jan_Temp_C = TempConvertF(Jan_Temp);
run;

proc print;
var Jan_Temp Jan_Temp_C;
run;
```



```
                           Jan_
Obs     Jan_Temp         Temp_C

 1        50.4          10.2222
 2        53.5          11.9444
 3        48.8           9.3333
 4        49.7           9.8333
 5        55.1          12.8333
```

PROC FCMP example

- Create a function that converts temperature from Fahrenheit to Celsius ***or*** Celsius to Fahrenheit



```
proc fcmp outlib=work.funcs.exampleC;
function TempConvert(temp, type $);
    if type = 'F' then
        return((temp-32)/1.8);
    else if type = 'C' then
        return ((temp*1.8)+32);
    endsub;
quit;
```

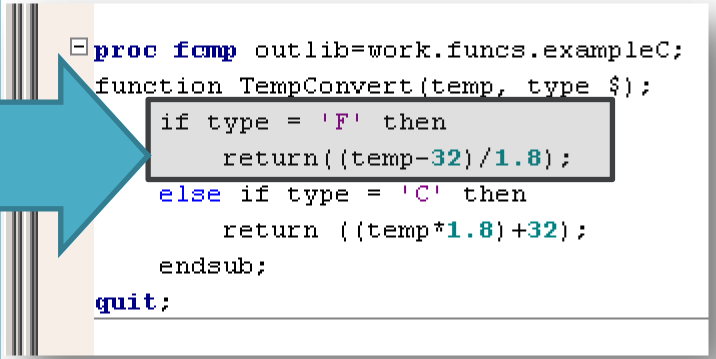
The new function **`TempConvert`** will have two arguments: **`temp`** and **`type`**

Type is a character variable

PROC FCMP example

- Create a function that converts temperature from Fahrenheit to Celsius ***or*** Celsius to Fahrenheit

If type is F (indicating the source temp is in Fahrenheit), **`TempConvert`** will return the temperature in Celsius

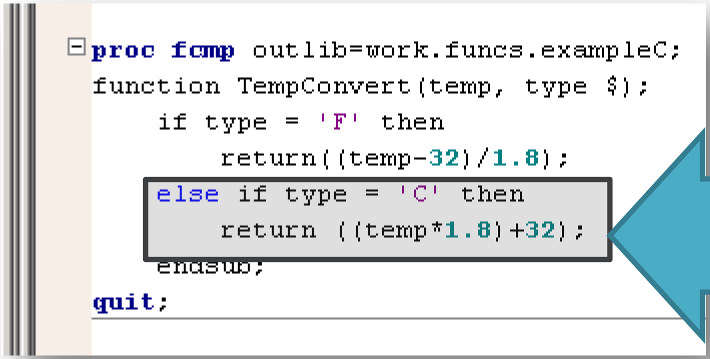


```
proc fcmp outlib=work.funcs.exampleC;
function TempConvert(temp, type $);
    if type = 'F' then
        return((temp-32)/1.8);
    else if type = 'C' then
        return ((temp*1.8)+32);
    endsub;
quit;
```

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP example

- Create a function that converts temperature from Fahrenheit to Celsius ***or*** Celsius to Fahrenheit

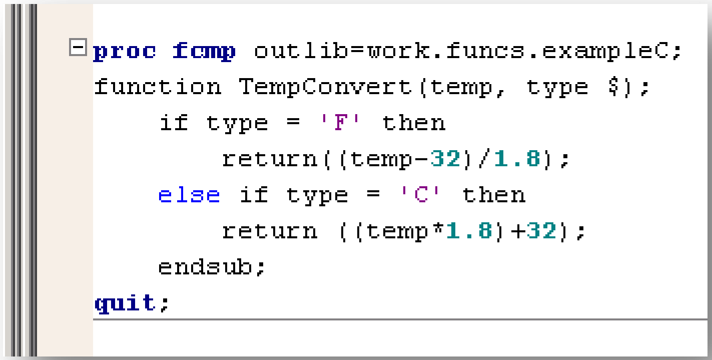


```
proc fcmp outlib=work.funcs.exampleC;
function TempConvert(temp, type $);
     if type = 'F' then
          return((temp-32)/1.8);
     else if type = 'C' then
          return ((temp*1.8)+32);
     endsub;
quit;
```

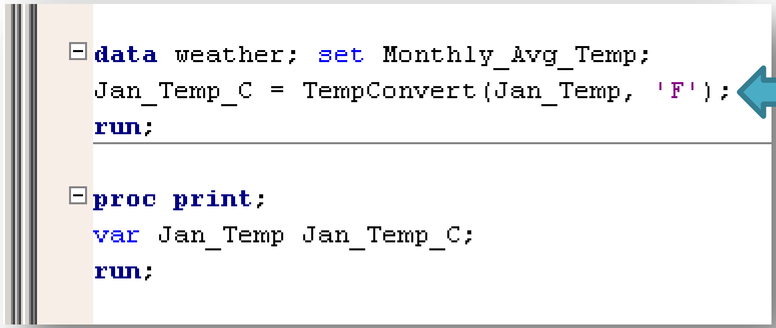
If type is C (indicating the source temp is in Celsius), **`TempConvert`** will return the temperature in Fahrenheit

PROC FCMP example

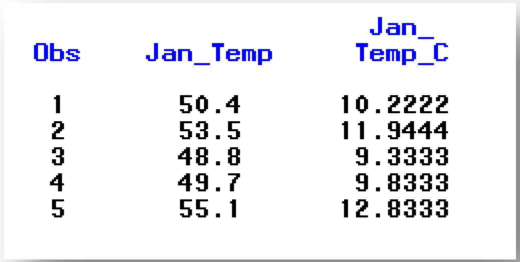
- Create a function that converts temperature from Fahrenheit to Celsius *or* Celsius to Fahrenheit



```
proc fcmp outlib=work.funcs.exampleC;
function TempConvert(temp, type $);
    if type = 'F' then
        return((temp-32)/1.8);
    else if type = 'C' then
        return ((temp*1.8)+32);
    endsub;
quit;
```



```
data weather; set Monthly_Avg_Temp;
Jan_Temp_C = TempConvert(Jan_Temp, 'F');
run;

proc print;
var Jan_Temp Jan_Temp_C;
run;
```

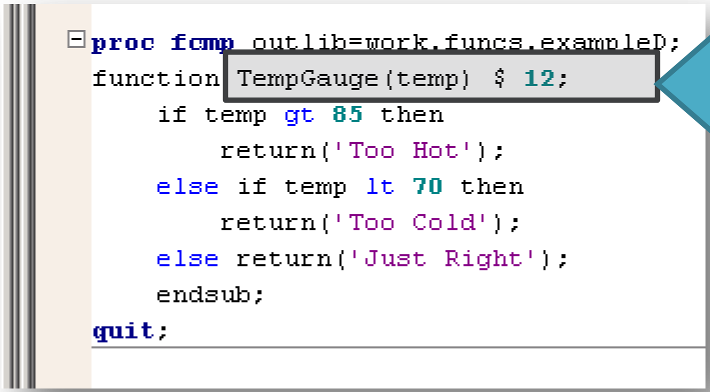


```
                          Jan_
 Obs      Jan_Temp       Temp_C

  1         50.4        10.2222
  2         53.5        11.9444
  3         48.8         9.3333
  4         49.7         9.8333
  5         55.1        12.8333
```

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP example

- Create a function that classifies the temperature as “Too Hot”, “Too Cold” or “Just Right”



```
proc fcmp outlib=work.funcs.exampleD;
function TempGauge(temp) $ 12;
    if temp gt 85 then
        return('Too Hot');
    else if temp lt 70 then
        return('Too Cold');
    else return('Just Right');
    endsub;
quit;
```

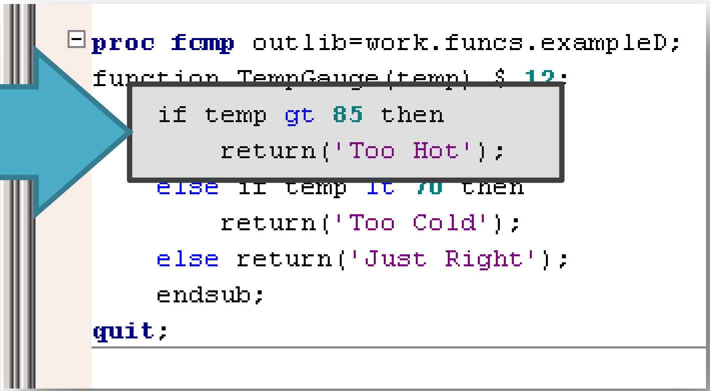
The new function **`TempGauge`** will have one argument: **`temp`**

The output value will be character with a length of 12

PROC FCMP example

- Create a function that classifies the temperature as “Too Hot”, “Too Cold” or “Just Right”

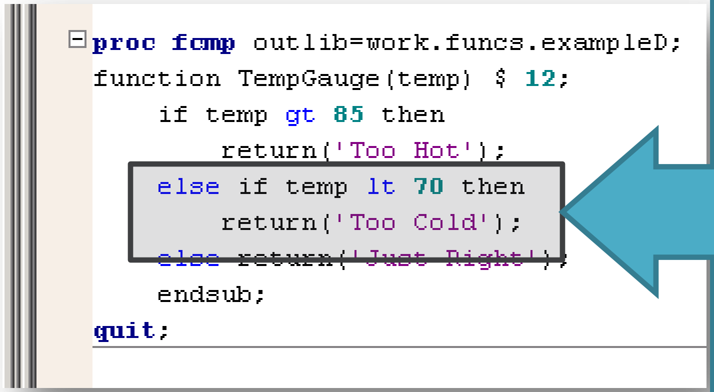
If the temp is greater than 85, **`TempGauge`** will return “Too Hot”



```
proc fcmp outlib=work.funcs.exampleD;
function TempGauge(temp) $ 12;
    if temp gt 85 then
        return('Too Hot');
    else if temp lt 70 then
        return('Too Cold');
    else return('Just Right');
    endsub;
quit;
```

PROC FCMP example

- Create a function that classifies the temperature as “Too Hot”, “Too Cold” or “Just Right”



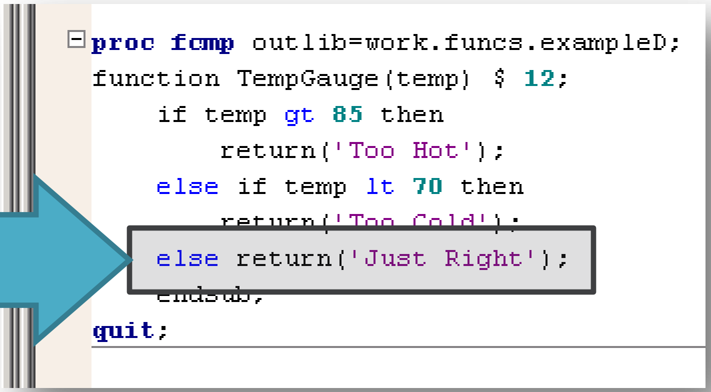
```
proc fcmp outlib=work.funcs.exampleD;
function TempGauge(temp) $ 12;
     if temp gt 85 then
          return('Too Hot');
     else if temp lt 70 then
          return('Too Cold');
     else return('Just Right');
     endsub;
quit;
```

If the temp is less than 70, **`TempGauge`** will return “Too Cold”

TATA CONSULTANCY SERVICES
Experience certainty.

PROC FCMP example

- Create a function that classifies the temperature as “Too Hot”, “Too Cold” or “Just Right”

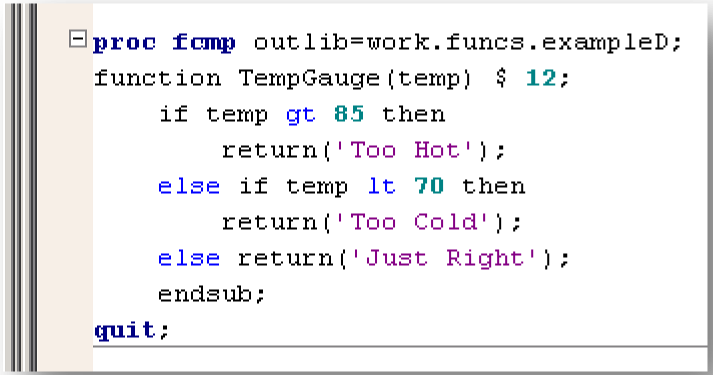


```
proc fcmp outlib=work.funcs.exampleD;
function TempGauge(temp) $ 12;
    if temp gt 85 then
        return('Too Hot');
    else if temp lt 70 then
        return('Too Cold');
    else return('Just Right');
    endsub;
quit;
```

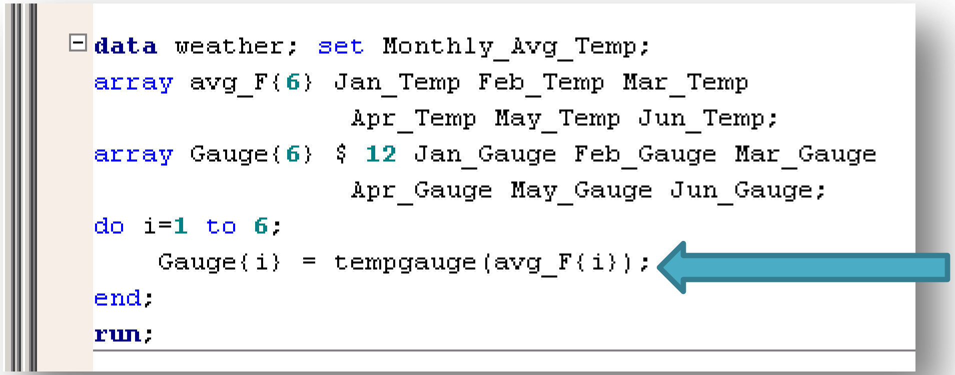
Otherwise, if the temp is between 70 and 85, **`TempGauge`** will return “Just Right”

PROC FCMP example

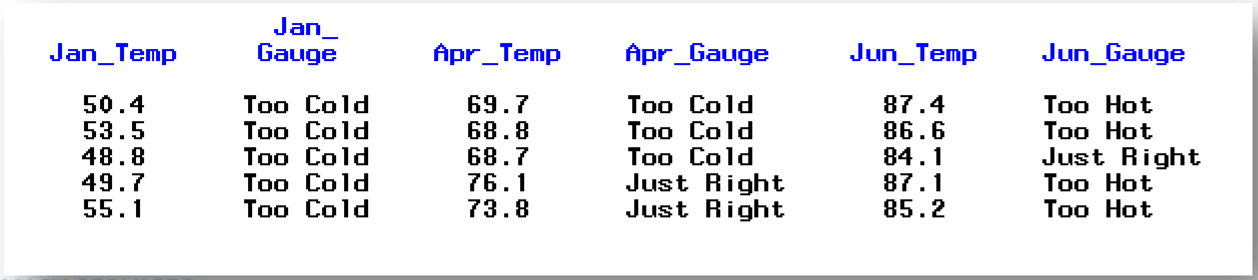
- Create a function that classifies the temperature as “Too Hot”, “Too Cold” or “Just Right”



```
proc fcmp outlib=work.funcs.exampleD;
function TempGauge(temp) $ 12;
     if temp gt 85 then
          return('Too Hot');
     else if temp lt 70 then
          return('Too Cold');
     else return('Just Right');
     endsub;
quit;
```



```
data weather; set Monthly_Avg_Temp;
array avg_F(6) Jan_Temp Feb_Temp Mar_Temp
                    Apr_Temp May_Temp Jun_Temp;
array Gauge(6) $ 12 Jan_Gauge Feb_Gauge Mar_Gauge
                    Apr_Gauge May_Gauge Jun_Gauge;
do i=1 to 6;
     Gauge(i) = tempgauge(avg_F(i));
end;
run;
```



```
                 Jan_
Jan_Temp         Gauge        Apr_Temp    Apr_Gauge     Jun_Temp    Jun_Gauge

  50.4        Too Cold          69.7     Too Cold         87.4     Too Hot
  53.5        Too Cold          68.8     Too Cold         86.6     Too Hot
  48.8        Too Cold          68.7     Too Cold         84.1     Just Right
  49.7        Too Cold          76.1     Just Right       87.1     Too Hot
  55.1        Too Cold          73.8     Just Right       85.2     Too Hot
```

Additional Reading

SAS Macro Programming for Beginners

Nine Steps to Get Started Using SAS Macros

SYMPUT and SYMGET: Getting DATA Step Variables and Macro Variables to Share

User-Written DATA Step Functions

A Cup of Coffee and Proc FCMP: I Cannot Function Without Them

Book: Carpenter's Complete Guide to the SAS Macro Language (available in the SOSUG Library)

Questions???