Ask the Expert Webinar Scenarios

Scenario 1- Creating and populating macro variables with values from a table column

Write a DATA step that reads the **ate.cards** table. When the value of **Row** is *25*, concatenate the values of **Card** and **Suit** with ' **of** ' in between and store the result in a macro variable named **MyCard**. For example, if **Card** was *Ace* and **Suit** was *Clubs*, the value of **MyCard** would be *Ace of Clubs*. The macro variable must be defined and populated in the DATA Step. After the DATA step executes, use macro statements to write the value of **MyCard** to the Log and to delete **MyCard** from the global symbol table.

Bonus:
Produce the value of **MyCard** using PROC SQL for a **Row** value of *13*.

Points to Ponder

- In a DATA step
  - An IF or WHERE statement could be used to choose the row with a **Row** value of 25. We'll use WHERE for efficiency's sake.
  - To write values into to a macro variable, we'll use the SYMPUTX call routine.
- In PROC SQL
  - We'll use the NOPRINT option to suppress the default report.
  - The INTO clause will write a value from the result into a macro variable.
  - The WHERE clause will select the desired row of data.
- We will use macro statements %PUT to write the value of **MyCard** to the Log and %SYMDEL to delete **MyCard** from the global symbol table.

Working the Problem

1. Open a new program window and enter the following code:

```
data _null_;
   set ate.cards;
   where Row=25;
   call symputx('MyCard',catx(' of ',Card,Suit));
run;

%put NOTE: &=MyCard;
%symdel MyCard;
```

2. Run the program and review the Log. Correct are re-run the program, if necessary, until there are no more errors or warnings in the Log.
3. The value of **MyCard** should be *Queen of Diamonds*.

Working the Bonus Problem

1. Open a new program window and enter the following code:

```
proc sql noprint;
   select catx(' of ',Card,Suit)
      into :MyCard
      from ate.cards
   where Row=13
;
quit;

%put NOTE: &=MyCard;
%symdel MyCard;
```

2. Run the program and review the Log. The value of **MyCard** should be *King of Clubs*.

Ask the Expert Webinar Scenarios

Scenario 2 - SQL SET operators

Using the set operators, return the name of the customers that have not responded to our 15OFF discount campaign.
Tables: **campaign**, **responded**, **customers**

- Create a query that returns the customer IDs that did not respond to the campaign.
- Use the query as a subquery to return the name of customer from the **customers** table

Bonus:

- Modify the query to display the name of the customers that did place an order using the 15OFF discount.

Working the Problem Step 1:

```
proc sql;
   select id from cert.campaign
      except
   select id from cert.responded
   ;
quit;
```

Working the Problem Step 2:

```
proc sql;
   select ID, Name
      from cert.customers
      where ID in (select ID from cert.campaign
                     except
                  select id from cert.responded)
   ;
quit
```

Working the Bonus Problem

```
proc sql;
   select ID, Name
      from cert.customers
      where ID in (select ID from cert.campaign
                     intersect
                  select id from cert.responded)
   ;
quit
```

Ask the Expert Webinar Scenarios

Scenario 3 - Why doesn't my Hash Object work?

My SAS program contains a hash object, but it won't run. It keeps giving me this error:

```
ERROR: Undeclared data symbol Type for hash object at line 233 column 10.
ERROR: DATA STEP Component Object failure.  Aborted during the EXECUTION phase.
```

Here's my SAS program:

```
/* Non-working code */
data certout.adoption;
   if _N_=1 then
      do;
         declare hash Adopt(dataset: 'cert.pets');
         adopt.definekey('ID');
         adopt.definedata('Type','Pet_Name');
         adopt.definedone();
      end;

   set cert.Names;
   if adopt.find() =0;
run;
```

Working the Problem

The **pets** table is not read in on a SET statement, so the necessary variables are not automatically created in the PDV. One solution is to create the variables ourselves with a LENGTH statement and initialize the values using the CALL MISSING statement.

```
data certout.adoption;
   length ID $4. Type $1. Pet_Name $20.;
   if _N_=1 then
      do;
         declare hash Adopt(dataset: 'cert.pets');
         adopt.definekey('ID');
         adopt.definedata('Type','Pet_Name');
         adopt.definedone();
         call missing(ID, Type, Pet_Name);
      end;

   set cert.Names;
   if adopt.find() =0;
run;
```

Another solution is to include a set statement for the **pets** table that doesn't execute. In that case, the compiler will create and initialize the variables in the PDV for us.

```
data certout.adoption;
   if 0 then
      set cert.pets;
   if _N_=1 then
      do;
         declare hash Adopt(dataset: 'cert.pets');
         adopt.definekey('ID');
```

```
          adopt.definedata('Type','Pet_Name');
          adopt.definedone();
       end;

   set cert.Names;
   if adopt.find() =0;
run;
```

Bonus Macro Scenario 1- Define and use a macro with parameters

Write a macro named **dealer** with two parameters, the first named **list** and the second named **item**. The macro should extract from **list** the item specified by **item**, ensure the value is in proper case, then store the value in a new global macro variable named **mycard**.

Call the macro **dealer** with the text *Ace, Deuce, Jack, King, Queen* as the value of list and 3 as the value of **item**. After the macro has finished processing, what is the value of **myCard**?

Points to Ponder

- We can use %SCAN to extract value from the list, and %SYSFUNC to execute the base SAS PROPCASE function to get the text into proper case.
- If the parameter values contain special characters, a macro quoting function may be needed.
- Macro variable **myCard** must be in the global symbol table to be available after macro execution ends.

Working the Problem

1. Open a new Code tab in SAS Studio.
   a. For our first try, enter the following code:

```
%macro dealer(list,item);
   %let myCard=%scan(&list,&item);
   %let myCard=%sysfunc(propcase(&myCard));
%mend dealer;

%dealer(CLUBS,diamonds,HEARTS,spades,3)
%put &=myCard;
```

   b. Run the program and review the Log. Note the error in the Log:

```
ERROR: More positional parameters found than defined.
```

   c. The first parameter contains commas, which are being misinterpreted as parameter delimiters. The macro failed to execute. Let's use %NRSTR to mask the commas when we call the macro.

2. Return to the Code Tab.
   a. For our second try, modify the code that calls the macro as follows:

```
%dealer(%nrstr(CLUBS,diamonds,HEARTS,spades),3)
%put &=myCard;
```

   b. Run the program and review the Log. Note the warning in the Log:

```
WARNING: Apparent symbolic reference MYCARD not resolved.
```

   c. Because macro variable **myCard** did not exist, it was created in the local symbol table. When the macro stopped executing, the loca symbol table was deleted and the variable myCard is no longer available. Let's use %GLOBAL to create **myCard** in the global symbol table.

3. Return to the Code Tab.
   a. To complete the program, modify the macro definition as follows:

```
%macro dealer(list,item);
   %global myCard;
   %let myCard=%scan(&list,&item);
   %let myCard=%sysfunc(propcase(&myCard));
%mend dealer;
```

   b. Run the program and review the Log. Verify there are no more warnings or errors. The value of **myCard** is **Hearts.**
   c. Because macro variable **myCard** did not exist, it was created in the local symbol table. When the macro stopped executing, the loca symbol table was deleted and the variable myCard is no longer available. Let's use %GLOBAL to create **myCard** in the global symbol table.

Bonus Macro Scenario 2- DO loop with fractional variable increments

Define a macro named **doLoop** that uses only macro statements to:

1. Create a global macro variable named **myVar3**.
2. Set the initial value of **myVar3** to 1.
3. Within a macro DO WHILE or DO UNTIL loop:
   a. Write the current value of **myVar3** to the SAS Log.
   b. Increment the value of **myVar3** by 0.25.
   c. Write the value of **myVar3** to the Log.
   d. When **myVar3 >**= 2, do not write the value to the log and stop processing the macro.
4. Call the macro and review the log to verify no errors or warnings.

Points to Ponder

- The macro facility only does integer math, so we will use the %SYSEVALF function to increment **myVar**.
- We will use a %DO %WHILE loop for the iterative processing.
- A %IF / %THEN statement can check if **myVar3 >**= 2.
- A %PUT statement will write the value of **myVar3** to the Log.
- A %RETURN statement will stop macro execution.

Ask the Expert Webinar Scenarios

Working the Problem

1. Open a new Code tab in SAS Studio.
    a. For our first try, enter the following code:

```
%macro doLoop;
   %local myVar;
   %let myVar=1;
   %do %while (&myvar<2);
      %let myVar=%sysevalf(&myVar+.25);
      %put &=myVar;
   %end;
%mend doLoop;
%doLoop
```

    b. Run the program and review the Log. Note the results:

```
MYVAR=1.25
MYVAR=1.5
MYVAR=1.75
MYVAR=2
```

    c. The process printed the value 2, which we didn't want output.

2. Return to the Code Tab.
    a. Insert a %IF statement to stop processing instead:

```
%macro doLoop;
   %local myVar;
   %let myVar=1;
   %do %while (&myvar<2);
      %let myVar=%sysevalf(&myVar+.25);
      %if &myVar >=2 %then %return;
      %put &=myVar;
   %end;
%mend doLoop;
%doLoop
```

    b. This excludes 2 from the results:

```
MYVAR=1.25
MYVAR=1.5
MYVAR=1.75
```