Beyond IF THEN ELSE: Techniques for Conditional Execution of SAS Code

Josh Horstman

Boston Area SAS Users Group (BASUG) Webinar

March 30, 2022

Introduction

- Conditional logic found in nearly every SAS program
- IF…THEN…ELSE is our primary tool
- Alternative methods can sometimes
  - Make our code more compact
  - Make our code more readable
  - Make our code easier to maintain or reuse

The SELECT Statement

Example #1: Original Code

```
if customer_type = 'STANDARD'
  then total = price * taxrate;

else if customer_type = 'PREFERRED'
  then total = price * discount * taxrate;

else if customer_type = 'TAXEXEMPT'
  then total = price;

else if customer_type = 'GOVERNMENT'
  then total = price * discount;
```

Example #1: Using the SELECT Statement

```
select(customer_type);
  when('STANDARD')    total = price * taxrate;
  when('PREFERRED')   total = price * discount * taxrate;
  when('TAXEXEMPT')   total = price;
  when('GOVERNMENT')  total = price * discount;
  otherwise total = price * taxrate;
end;
```

- Each when-expression compared against select-expression
- First true comparison → corresponding statement executed
- If none compare true, “otherwise” statement is executed
- No match + no otherwise statement = SAS error in the log

The IFC and IFN Functions

The IFC and IFN Functions

- IFC: Returns a character value based on whether an expression is true, false, or missing

- IFN: Returns a numeric value based on whether an expression is true, false, or missing

- IFN(logical_expression,
  value_returned_when_true,
  value_returned_when_false,
  <value_returned_when_missing>)

Example #1: Using the IFN Function

```
total = price *

   ifn(customer_type in ('PREFERRED','GOVERNMENT'),
       discount,
       1)
   *
   ifn(customer_type in ('STANDARD','PREFERRED'),
       taxrate,
       1);
```

Example #2: Original Code

```
if married='Y' and num_kids=0
  then family_status = 'Married, no children';

if married='N' and num_kids=0
  then family_status = 'Unmarried, no children';

if married='Y' and num_kids=1
  then family_status = 'Married, 1 child';

if married='N' and num_kids=1
  then family_status = 'Unmarried, 1 child';

if married='Y' and num_kids>1
  then family_status = 'Married, '||
    strip(put(num_kids,best.))||' children';

if married='N' and num_kids>1
  then family_status = 'Unmarried, '||
    strip(put(num_kids,best.))||' children';
```

Example #2: Streamlined with IFC

```
family_status = catx(' ',
    ifc(married='Y','Married,','Unmarried,'),
    ifc(num_kids=0,'no',put(num_kids,best.)),
    ifc(num_kids=1,'child','children'));
```

Example #3: Using IFC

- We want to format a percentage as follows:
  - Exactly 0 formatted as 0.0%
  - Between 0 and 0.1 formatted as <0.1%
  - 0.1 through 99.9 rounded to the nearest tenth (XX.X%)
  - Between 99.9 and 100 formatted as >99.9%
  - Exactly 100 formatted as 100%

Numerical Value	Formatted String
0	0.0%
0.05	<0.1%
98.7654321	98.8%
99.91	>99.9%
100	100%

- There are many ways to do this!
  - Series of IF/THEN/ELSE statements
  - SELECT/WHEN block
  - PROC FORMAT using PICTURE statement … almost
  - Nested IFC functions

Example #3: Using IFC

```
ifc( pctval = 100, '100',
   ifc( pctval > 99.9, '>99.9',
      ifc( pctval = 0, '0.0',
         ifc( pctval < 0.1,'<0.1',
            strip(put(round( pctval,0.1),4.1))))))
```

- It's an expression, not a complete statement
  - Use it as an argument to another function
  - Make it a macro and use it in-line

Example #3: Using IFC

```
%macro fmtpct(pctval);
   ifc(&pctval = 100, '100',
      ifc(&pctval > 99.9, '>99.9',
         ifc(&pctval = 0, '0.0',
            ifc(&pctval < 0.1,'<0.1',
               strip(put(round(&pctval,0.1),4.1))))))
%mend fmtpct;
```

- It's an expression, not a complete statement
  - Use it as an argument to another function
  - Make it a macro and use it in-line

```
CI = cats('(',%fmtpct(lcl),'-',%fmtpct(ucl),')');
```

A Note About IFN / IFC

```
data test1;
  x=0;
  if x ne 0 then y=10/x;
  else y=999;
run;

data test2;
  x=0;
  y = ifn(x ne 0, 10/x  999);
run;
```

IFN and IFC evaluate ALL arguments regardless of value of logical expression!

Replace with **DIVIDE(10,x)** to eliminate the log note.

```
NOTE: Division by zero detected at line 21 column 27.
x=0 y=999 _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following
places. The results of the operations have been set to missing
values.
Each place is given by: (Number of times) at (Line):(Column).
      1 at 21:27
```

PROC FORMAT

Example #4: The Setup

We have:

PLANETS1
PLANET_NAME
Venus
Neptune
Jupiter
Earth
Mars
Saturn
Uranus
Mercury

We want:

PLANETS2	
PLANET_NAME	PLANET_ORDER
Venus	2
Neptune	8
Jupiter	5
Earth	3
Mars	4
Saturn	6
Uranus	7
Mercury	1

Example #4: Original Code

```
data planets2;
  set planets1;
  if      upcase(planet_name) = 'MERCURY' then planet_order=1;
  else if upcase(planet_name) = 'VENUS'   then planet_order=2;
  else if upcase(planet_name) = 'EARTH'   then planet_order=3;
  else if upcase(planet_name) = 'MARS'    then planet_order=4;
  else if upcase(planet_name) = 'JUPITER' then planet_order=5;
  else if upcase(planet_name) = 'SATURN'  then planet_order=6;
  else if upcase(planet_name) = 'URANUS'  then planet_order=7;
  else if upcase(planet_name) = 'NEPTUNE' then planet_order=8;
run;
```

Example #4: Custom Format

```
proc format;
  invalue planets
    'MERCURY' = 1
    'VENUS'   = 2
    'EARTH'   = 3
    'MARS'    = 4
    'JUPITER' = 5
    'SATURN'  = 6
    'URANUS'  = 7
    'NEPTUNE' = 8;
 run;

 data planets2;
   set planets1;
   planet_order = input(upcase(planet_name),planets.);
 run;
```

The WHICHC/WHICHN and CHOOSEC/CHOOSEN Functions

The WHICHC and WHICHN Functions

- Introduced in SAS 9.2
- Search for a value equal to first argument, return index of the first matching value
- WHICHC(value_to_find, arg1, arg2, …)
- WHICHC: arguments are character values
  WHICHN: arguments are numeric values
- Both return a numeric value

Example #4 with WHICHC

```
data planets2;
 set planets1;
 planet_order =
    whichc(planet_name,'MERCURY',
     'VENUS','EARTH','MARS','JUPITER',
     'SATURN','URANUS','NEPTUNE');
run;
```

Example #5: Using WHICHN

- Use WHICHC when concatenating datasets to create a variable indicating the source for each record.

```
data all;
  set ds1(in=in1)
      ds2(in=in2)
      ds3(in=in3);
  source = whichn(1,in1,in2,in3);
run;
```

- SOURCE will contain a 1 for records from DS1, etc.

Example #6: Using WHICHC

```
proc summary data=sashelp.cars;
   var mpg_city;
   output out=carsumm
      (drop=_TYPE_ _FREQ_);
run;


data carsumm2;
   set carsumm;
   rowsort =
     whichc(_STAT_,'N','MEAN','STD','MIN','MAX');
run;
```

STAT	MPG_CITY	ROWSORT
N	428	1
MIN	10	4
MAX	60	5
MEAN	20.06	2
STD	5.24	3

Create a custom sorting variable with WHICHC.

The CHOOSEC and CHOOSEN Functions

- Introduced in SAS 9.0
- Return a value that represents the results of choosing from a list of arguments
- CHOOSEC(index,choice1,choice2,…)
- CHOOSEC: chooses from and returns a character value
  CHOOSEN: chooses from are returns a numeric value
- Index is always numeric

Example #7: Using CHOOSEN

- Combine CHOOSEN with WEEKDAY and HMS to get today's closing time as a SAS time value.

```
data choosen_example;
   closing_time =
    hms(choosen(weekday(today()),
                17,20,20,20,20,21,22),0,0);
   format closing_time timeampm8.;
   put closing_time;
run;
```

The value returned by WEEKDAY serves as the index for CHOOSEN.

The value returned by CHOOSEN is the HOUR argument to HMS.

SAS Output:

```
8:00 PM
```

Example #8: The Setup

We have:

RESPONSE1
RESPONSE_CODE
PD
PR
CR
SD
NE

We want:

RESPONSE2	
RESPONSE_CODE	RESPONSE
PD	Progressive Disease
PR	Partial Response
CR	Complete Response
SD	Stable Disease
NE	Not Evaluable

Example #8 – Combining CHOOSEC and WHICHC

```
data response2;
    set response1;
    response = choosec(
       whichc(response_code,'PD','PR','CR','SD','NE'),
        'Progressive Disease',
        'Partial Response',
        'Complete Response',
        'Stable Disease',
        'Not Evaluable');
run;
```

WHICHC returns a number 1 through 5 which is passed as the first parameter to CHOOSEC

The COALESCE and COALESCEC Functions

Example #9: The Setup

- Goal: Derive date of last contact (LSCONDT)
- Algorithm:
  - Use date of death (DTHDT) if present
  - Otherwise use date of withdrawal (WDDT) if present
  - Otherwise use date of last visit (LSTVISDT) if present
  - Otherwise use date of last dose (LSTDOSDT) if present

Example #9: Original Code

```
if not missing(dthdt)
 then lstcondt = dthdt;

else if not missing(wddt)
 then lstcondt = wddt;

else if not missing(lstvisdt)
 then lstcondt = lstvisdt;

else lstcondt = lstdosdt;
```

Example #9: Using the COALESCE and COALESCEC Functions

- COALESCE / COALESCEC returns the first argument that is not missing

- Use COALESCE for numeric data
  Use COALESCEC for character data

```
lstcondt = coalesce(dthdt, wddt, lstvisdt, lstdosdt);
```

Comparison Operators

Example #10: The Setup

- Want to derive study day
  - Date of first dose is Day 1
  - Day prior to first dose is Day -1

- We have:

EXSTDT	AESTDT
01MAY2016	10MAY2016
01MAY2016	26APR2016

- We want:

EXSTDT	AESTDT	AESTDY
01MAY2016	10MAY2016	10
01MAY2016	26APR2016	-5

Example #10: Original Code

```
data example10a;
  set example10;
  if aestdt >= exstdt
  then aestdy = aestdt - exstdt + 1;
  else aestdy = aestdt - exstdt;
run;
```

Example #10: Using IFN

```
data example10b;
    set example10;
    aestdy = aestdt - exstdt
                + ifn(aestdt >= exstdt,1,0);
run;
```

Example #10: Using a Comparison Operator

```
data example10c;
    set example10;
    aestdy = aestdt - exstdt + (aestdt >= exstdt);
run;
```

The Subsetting IF Statement

The Subsetting IF

- An IF with no THEN
- Continues processing only those records meeting the condition
- If condition not met, current observation not written to data set
- Functionally equivalent to: `if not <expression> then delete;`

The Subsetting IF

THIS...

```
data demog2;
   set demog;
   if not nmiss(height,weight);
   bmi = weight / (height**2);
run;
```

IS EQUIVALENT TO THIS...

```
data demog2;
   set demog;
   if not (not nmiss(height,weight)) then delete;
   bmi = weight / (height**2);
run;
```

Subsetting IF vs. WHERE Statement

WHERE statement

- Applied BEFORE observation read into PDV*
- Can be used in many SAS procedures
- Possible efficiency improvements
- Non-executable statement
- Always applied at beginning of DATA step, regardless of where statement appears
- Can use special operators such as CONTAINS, LIKE, and BETWEEN/AND
- Can only access variables from input data set(s)

Subsetting IF statement

- Applied AFTER observation read into PDV*
- Only used in the DATA step
- Reads and processes every record
- Executable statement
- Can be applied at any point in the DATA step, depending on where statement appears
- Can use automatic variables such as FIRST.BY, LAST.BY, and _N_
- Can use newly created variables

* PDV = Program Data Vector

Subsetting IF vs. WHERE Statement

ITEMS1

ITEM	QUANTITY
X	17
Y	15
Z	18

ITEMS2

ITEM	QUANTITY
X	19
Y	21
Z	23

```
data items_where;
   merge items1 items2;
   by item;
   where quantity < 20;
run;
```

ITEMS_WHERE

ITEM	QUANTITY
X	19
Y	15
Z	18

```
data items_if;
   merge items1 items2;
   by item;
   if quantity < 20;
run;
```

ITEMS_IF

ITEM	QUANTITY
X	19

The %IF…%THEN…%ELSE Macro Statement

%IF…%THEN…%ELSE Macro Statements

- Similar logic to IF…THEN…ELSE but operates on a macro level

- IF…THEN…ELSE used to conditionally execute code
  %IF…%THEN…%ELSE used to conditionally generate code

Using the %IF Macro Statement

- Unlike the IF statement, the %IF macro statement can conditionally include entire procedure calls

```
%MACRO do_stuff(mydset,printyn,freqyn);

   %IF &printyn = Y %THEN %DO;
      proc print data=&mydset; run;
   %END;

   %IF &freqyn = Y %THEN %DO;
      proc freq data=&mydset; run;
   %END;

%MEND do_stuff;
```

You can't use IF statements here!

Calling

```
%do_stuff(demog,Y,Y)
```

generates this code:

```
proc print data=demog; run;
proc freq data=demog; run;
```

Conditionally Generating Conditional Code

```
%MACRO merge_demog(dset1,dset2,addbmiyn);

proc sort data=&dset1; by subject; run;
proc sort data=&dset2; by subject; run;

data demog;
   merge &dset1 &dset2;
   by subject;
   %IF &addbmiyn = Y %THEN %DO;
      if not nmiss(height,weight) then
         bmi = weight / (height**2);
   %END;
run;

%MEND merge_demog;
```

This macro accepts three parameters and generates two PROC SORTs and a DATA step.

This IF statement is included in the DATA step only when the value of the macro parameter ADDBMIYN is Y.

Conclusions

Conclusions

- SAS provides many ways to implement conditional logic.
- The best programmers have many tools at their disposal.
- Learn how and when to use each one.

Any Questions?

Contact Information:

Joshua M. Horstman
Nested Loop Consulting
josh@nestedloopconsulting.com