Different Decimal Places For Different Lab Tests

A. Cecilia Mauldin, PPD Development, Morrisville, NC.

ABSTRACT
This paper will show how to present the value of one variable in multiple formats. The representation of a variable can depend on the value of a second variable or in the value of the observation itself.

In this case, the representation of the results of a laboratory test will depend on the name of the laboratory test. The format used for each laboratory test is independent from other laboratory tests.

Each laboratory test can have its own appropriate number of significant decimal places and the number can vary from test to test. This method guarantees that significant results are not truncated, and also, only significant decimals are presented.

Through this paper, a series of examples will show how to use the PUTN function with two, three or four parameters, and the difference between rounding or not rounding before formatting.

INTRODUCTION
When presenting results from laboratory tests, it is desired that the format of a result (or the statistics) of each laboratory test is independent from other laboratory tests.

This independence can be achieved either by creating a format for each of the laboratory tests. Using the function PUTN can be elegant and fast, because it involves one line of code compared to many IF statements.

Following is the data that will be used in these examples. The dataset LABRESLT has the results received from a lab, a character variable with the laboratory name and the results. The dataset USEINFOR is a dataset that can be by the programmer if he/she knows what formats are needed for each test. The USEINFOR dataset has one record for each laboratory test, two variables that can be used as formats and a variable that can be used to round.

data labreslt;
  input Lab $ Result @@;
cards;
  HCT  35.36    HCT  35.046
  PLT  118.045  PLT  118.111 PLT  118.666
  RDW  16.16    RDW  17.166  RDW  12.1666 RDW  12.16666
  WBC  4.0      WBC  10.005
;run;

data useinfor;
  input lab $ frstf $ secf $ rnd @@;
cards;
  HCT  5.1 8.0 0.1
  PLT  5.0 8.4 0.01
  RDW  5.2 8.6 0.1
  WBC  7.4 8.8 0.001
;run;

data toprint;
merge labreslt useinfor;
by lab;
run;

proc print data=toprint;
by lab;
id lab;
run;

DATASET TOPRINT

<table>
<thead>
<tr>
<th>LAB</th>
<th>RESULT</th>
<th>frstf</th>
<th>secf</th>
<th>rnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCT</td>
<td>35.360</td>
<td>5.1</td>
<td>8.0</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>35.046</td>
<td>5.1</td>
<td>8.0</td>
<td>0.100</td>
</tr>
<tr>
<td>PLT</td>
<td>118.045</td>
<td>5.0</td>
<td>8.4</td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td>118.111</td>
<td>5.0</td>
<td>8.4</td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td>118.666</td>
<td>5.0</td>
<td>8.4</td>
<td>0.010</td>
</tr>
<tr>
<td>RDW</td>
<td>16.160</td>
<td>5.2</td>
<td>8.6</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>17.166</td>
<td>5.2</td>
<td>8.6</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>12.167</td>
<td>5.2</td>
<td>8.6</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>12.167</td>
<td>5.2</td>
<td>8.6</td>
<td>0.100</td>
</tr>
<tr>
<td>WBC</td>
<td>4.000</td>
<td>7.4</td>
<td>8.8</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>10.005</td>
<td>7.4</td>
<td>8.8</td>
<td>0.001</td>
</tr>
</tbody>
</table>

The variable, RESULT, is printed with the default format for all laboratory values.

**Example 1: Using PUTN with a variable that already exists**

In this example, the programmer provides the formats for different observation in separate variables. The character variables FRSTF and SECF already exist in the dataset TOPRINT. Also, as we have seen before, FRSTF and SECF are expressions which have values used as numeric formats.

data firste;
  set toprint;
  formatm=putn(result,frstf,22,3);
  std=putn(result,secf);
run;

proc print data=firste split='*';
by lab;
id lab;
var result frstf formatm secf std;
label formatm="FORMATM: result with format given by FRSTF"
  std="STD: result with format given by SECF ";
run;

The first parameter, which is passed to the function PUTN, is the variable that will be formatted, in this case the variable RESULT. The second parameter is a character variable that contains a numeric format that will be applied, in this example FRSTF and SECF.
This numeric format will be applied to the variable specified in the first parameter unless there is a third and/or fourth parameter that overwrite the width and the number of decimal places to be used.
The created variables `FORMATM` and `STD` are character variables. The value of `FORMATM` is the character expression of `RESULT` with format=22.3 for all observations. The value for `STD` is the value given by `SECF`.

<table>
<thead>
<tr>
<th>LAB</th>
<th>VALUE</th>
<th>frstf</th>
<th>format 22.3</th>
<th>secf</th>
<th>format given by SECF</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCT</td>
<td>35.360</td>
<td>5.1</td>
<td>35.360</td>
<td>8.0</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>35.046</td>
<td>5.1</td>
<td>35.046</td>
<td>8.0</td>
<td>35</td>
</tr>
<tr>
<td>PLT</td>
<td>118.045</td>
<td>5.0</td>
<td>118.045</td>
<td>8.4</td>
<td>118.0450</td>
</tr>
<tr>
<td></td>
<td>118.111</td>
<td>5.0</td>
<td>118.111</td>
<td>8.4</td>
<td>118.1110</td>
</tr>
<tr>
<td></td>
<td>118.668</td>
<td>5.0</td>
<td>118.666</td>
<td>8.4</td>
<td>118.6660</td>
</tr>
<tr>
<td>RDW</td>
<td>16.160</td>
<td>5.2</td>
<td>16.160</td>
<td>8.6</td>
<td>16.16000</td>
</tr>
<tr>
<td></td>
<td>17.166</td>
<td>5.2</td>
<td>17.166</td>
<td>8.6</td>
<td>17.16600</td>
</tr>
<tr>
<td></td>
<td>12.167</td>
<td>5.2</td>
<td>12.167</td>
<td>8.6</td>
<td>12.16660</td>
</tr>
</tbody>
</table>

The observations for `RDW` were supposed to be formatted as 8.6 but because the significant number of spaces to express the given number of integers needed only allowed for 5 decimal to be displayed. It is important to give a width that is big enough to display integers and decimals.

**Example2: Calculating the format**

It is desired that the variable that gives the format is not only a constant but a value that can be modified. This modification can depend on the value of the `RESULT` variable or the formatting variable itself. It is possible to use different formats when the results are out of the Normal Limits.

```sas
data seconde;
  set toprint;
  if input(frstf,best.)> 4.1 then
    meanf=putn(result,put(input(frstf,best.)-1.1,best.));
  else
    meanf=putn(result,put(input(secf,best.),best.));
run;

proc print data=seconde split='*';
  by lab;
  id lab;
  var result frstf secf meanf;
  label meanf="MEANF: Result formated as frstf - 1.1 when frstf>4.1";
run;
```

What happened to the variable `MEANF` when `LAB='PLT'`?

You can't have a digit in the decimal place that is larger than the digit in the integer place. 5.0 – 1.1 is 3.9! You can not represent a numeric variable with nine decimal places with a width of 3. Again, it is important to know the data when applying a format. Or know how many decimals are needed to represent a value. In some cases, it is desired to round a result.
Example 3: Using the ROUND function

This example shows how to use the **ROUND** function using a variable as the second parameter.

```plaintext
data thirde;
  set toprint;
  raw=round(result,rnd);
run;
```

```plaintext
proc print data=thirde (obs=6);
  by lab;
  id lab;
  var result rnd raw;
run;
```

In the observations where variable **RND** equals 0.10, the variable **RAW** has one significant decimal place, whether the original value has one or more significant decimal values. In the observations where **RND**=.01, the variable **RAW** has two significant decimal places.

Example 4: Formatting the data before or after rounding

This example will show the difference in the results when a variable is formatted before or after rounding.
data four;
set toprint;

c_round=round(result,.01);
w_round=round(result,rnd);
after_round=putn(c_round,secf);
no_round=putn(result,secf);
run;

proc print data=four width=min split='*';
by lab;
id lab;
var result rnd secf c_round w_round after_round no_round;
run;

<table>
<thead>
<tr>
<th>LAB</th>
<th>result</th>
<th>rnd</th>
<th>secf</th>
<th>c_round</th>
<th>w_round</th>
<th>after_round</th>
<th>no_round</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCT</td>
<td>35.360</td>
<td>0.100</td>
<td>8.0</td>
<td>35.36</td>
<td>35.400</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>35.046</td>
<td>0.100</td>
<td>8.0</td>
<td>35.05</td>
<td>35.000</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>PLT</td>
<td>118.045</td>
<td>0.010</td>
<td>8.4</td>
<td>118.05</td>
<td>118.050</td>
<td>118.0500</td>
<td>118.045</td>
</tr>
<tr>
<td></td>
<td>118.111</td>
<td>0.010</td>
<td>8.4</td>
<td>118.11</td>
<td>118.110</td>
<td>118.1100</td>
<td>118.111</td>
</tr>
<tr>
<td></td>
<td>118.666</td>
<td>0.010</td>
<td>8.4</td>
<td>118.67</td>
<td>118.670</td>
<td>118.6700</td>
<td>118.666</td>
</tr>
<tr>
<td>RDW</td>
<td>16.160</td>
<td>0.100</td>
<td>8.6</td>
<td>16.16</td>
<td>16.200</td>
<td>16.16000</td>
<td>16.1600</td>
</tr>
<tr>
<td></td>
<td>17.166</td>
<td>0.100</td>
<td>8.6</td>
<td>17.17</td>
<td>17.200</td>
<td>17.17000</td>
<td>17.1660</td>
</tr>
<tr>
<td></td>
<td>12.167</td>
<td>0.100</td>
<td>8.6</td>
<td>12.17</td>
<td>12.200</td>
<td>12.17000</td>
<td>12.1660</td>
</tr>
<tr>
<td></td>
<td>12.167</td>
<td>0.100</td>
<td>8.6</td>
<td>12.17</td>
<td>12.200</td>
<td>12.17000</td>
<td>12.1660</td>
</tr>
<tr>
<td>WBC</td>
<td>4.000</td>
<td>0.001</td>
<td>8.8</td>
<td>4.00</td>
<td>4.000</td>
<td>4.000000</td>
<td>4.00000</td>
</tr>
<tr>
<td></td>
<td>10.005</td>
<td>0.001</td>
<td>8.8</td>
<td>10.01</td>
<td>10.005</td>
<td>10.01000</td>
<td>10.0050</td>
</tr>
</tbody>
</table>

For most observations there is a difference between the variables AFTER_ROUND and NO_ROUND in both cases, the variable was formatted using the variable SECF.
The difference between rounding and formatting with a variable and a constant appears if you want different rounding and formatting for different observations.

Should you format before or after? Round the variable to the desired number of decimals first, and then format the variable to the desired representation.

Example 5: Making the data guide its own format

It is good to get the format for a variable from an outside source, but it is convenient to have the data provide information about the format that will better fit itself. The maximum number of decimals in for each laboratory test is one of the best guides to know what is significant for the test.

This example will show how to represent the mean and the standard deviation for the results of the dataset LABRSLT with respectively one decimal more and two decimal places more than what the results have.

PROC SQL;
CREATE TABLE LABS AS SELECT DISTINCT LAB,
RESULT,
PUT(result,BEST.) AS CHAR
FROM LABRSLT
ORDER BY lab
;
The dataset LABS has all different results and the results of the dataset LABRSLT. Using the format BEST, it is guaranteed that the variable CHAR will keep all the decimals that were given by the data.

The variables in LABS ARE:
LAB, THE LABORATORY TEST NAME.
RESULT THE NUMERIC RESULT.
CHAR, THE CHARACTER RESULT.

CREATE TABLE DECIMAL AS
SELECT DISTINCT
LAB,
MAX(LENGTH(CHAR)-INDEX(CHAR,'.')) AS DECIMAL
FROM LABS
WHERE CHAR LIKE '%.%'
GROUP BY LAB;

The dataset DECIMAL has one record per laboratory test where a decimal place is used. The variable DECIMAL has the value of the maximum number of decimals used in each test. This method can be used to check that the labs have the number of decimals that is expected.

SELECT * FROM DECIMAL;

The dataset DECIMAL has a record for each laboratory test in the LABS dataset. The variable DECIMAL has the maximum number of decimal places for each laboratory test.

CREATE TABLE DECIMAL AS
SELECT DISTINCT
LAB,
DECIMAL,
10**(-DECIMAL -1) FORMAT BEST. AS ONERND,
10**(-DECIMAL -2) FORMAT BEST. AS TWORND,
"22." ||PUT(DECIMAL +1,1.) AS ONEFMT,
"22." ||PUT(DECIMAL +2,1.) AS TWOFMT
FROM DECIMAL;

SELECT * FROM DECIMAL;

When an observation in the dataset DECIMAL has N significant decimal places, the variable ONERND shows a value with N+1 decimal places and the variable TWORND shows a value of N+2 decimal places. The variables ONEFMT and TWOFMT show a value that can be used as a format with a width of 22 and one and two more decimal places respectively.

QUIT;

To calculate statistics on the variable RESULT for each of the LAB tests:

PROC UNIVARIATE DATA=LABRSLT NOPRINT;

VAR RESULT;
BY LAB;

OUTPUT OUT=STAT MEAN=MEAN STD=STD;
RUN;

PROC PRINT DATA=STAT;
RUN;

DATA FORMATED;
MERGE STAT DECIMAL;
BY LAB;

RMEAN=ROUND(MEAN,ONERND);
FMEAN=PUTN(RMEAN,ONEFMT);
FSTD=PUTN(ROUND(STD,TWORND),TWOFMT);
RUN;

PROC PRINT DATA=FORMATED WIDTH=MIN;
VAR LAB MEAN STD DECIMAL RMEAN FMEAN FSTD;
RUN;

FINAL COMMENTS
The second parameter passed to the PUTN function is a character variable, this allows integer formats that are not only integers like DATE8., MMDDYY10. to be passed to the function. This presents the disadvantage that any arithmetic that is done in the data step section needs to be converted to character to avoid having error or warning messages in the LOG file.

It may be easier to keep the second parameter without variation and overwrite the width and the number of decimal places with a third and fourth parameters, which can be numeric. Because the conversion of arithmetic expressions to character gets tiresome if a third or fourth parameters are not used. However, it can be confusing to have a parameter that is meaningless.

CONCLUSION
The function PUTN allows variables to have different formats without the use of many IF statements, the formats used can depend in the data used and can help find problematic data.

ACKNOWLEDGEMENTS
I want to thank Craig Mauldin, Dan Sharp, Douglas Criger, and Jennifer Bridgers, who read the paper and made very good comments, and Bonnie Duncan and Kim Sturgen for their encouragement.

REFERENCES

CONTACT INFORMATION
A. Cecilia Mauldin
PPD Development
3900 Paramount Parkway
Morrisville, NC 27560
(919) 462-4199
Cecilia.Mauldin@rtp.ppd.com