Those Sneaky SAS® Functions: Beware of Unexpected Handling of Missing Data and Variable Lists
Christianna S. Williams, University of North Carolina at Chapel Hill, Chapel Hill, NC

ABSTRACT
When is a SUM not a sum? When is a hyphen not a hyphen? SAS® arithmetic and summary functions don’t always behave the way you might expect when it comes to missing values and variable lists. In this brief presentation, I’ll confess to how I’ve gotten tripped up by making assumptions about the behavior of a few commonly used functions and perhaps save you from making similar mistakes.

INTRODUCTION
At last count, there were 473 SAS functions and CALL routines listed in the Base SAS documentation. (This should keep me busy writing Coder’s Corner papers for a while…) Among the ones that I use the most, and the subject of this paper, are just a few of the twenty-odd functions in the category “Descriptive statistics”. These may not be the most glamorous of SAS functions, but they are very useful – and not just in “statistical” applications. However, as with many things in life, it’s important to read the fine print. I’ve discovered – sometimes the hard way – that these seemingly very straightforward functions do not always behave the way you might first think. In particular, I’m going to explain two caveats with respect to the behavior of these functions. The first has to do with the treatment of missing values in the argument list, and the second has to do with the use of shorthand variable lists as arguments to the functions. I’m going to share a few lessons learned – for the SAS programmer just starting to use functions and variable lists as arguments to functions. All of the examples in this paper start with the data set OODLES, shown in Figure 1.

LEARNING HOW TO ADD
When I first started SAS programming too many moons ago, I was completely unaware that many simple arithmetic and basic statistical tasks could be accomplished within the DATA Step using SAS Functions. For example, if I had the data set OODLES, shown in Figure 1, and I wanted to get a sum of the noodle variables, I would have used the following statement:

```sas
noodlesum = noodle1 + noodle2 + noodle3;
```

This is not too much typing, and works just fine, giving me a missing value for noodlesum in the three cases where one or more of the noodle variables is missing (IDs E, I and J), which is what I would expect. (I’m not sure if this is what everyone would expect, but it makes sense to me that if one or more of the addends in a sum is unknown then the sum will be unknown). And, of course this same sort of syntax works if I want to add variables with names that are not in the X1, X2, …, Xn format, such as

```sas
sillysum = doodle1 + noodle1 + strudel1;
```

But then I saw a coworker’s program, and saw she had done seemingly the same thing in a manner that seemed much cooler to me (She had some noodle):

```sas
sumnoodle = SUM(noodle1, noodle2, noodle3);
```

Of course, I probably didn’t look in the SAS manual (and Ron Cody’s function book was not to be published for another decade or more!), and even if I had, I might have blithely overlooked a certain adjective in the description of the SUM function:
“Returns the sum of the nonmissing arguments” (my italics). So, I was surprised when I discovered that instead of having three missing values for my sum, I now had only one (See Figure 2). What's going on?

Well, basically, the SUM function ignores the missing value, which in effect treats it as a zero (unless all the arguments are missing – as in observation J – in which case they are not treated as 0 but instead as missing values), which I have to say, still seems kind of strange to me, but I've gotten used to it. Note in particular that observations D and E are identical for the noodle variables except that in observation D noodle 1 is 0 and in observation E noodle 1 is missing. They yield different values for noodlesum but the same when the SUM function is used (sumnoodle). Sometimes it's useful for missing values to be ignored in this way, but if it isn't you can use another useful function to get the same values as noodlesum:

```plaintext
IF N(noodle1, noodle2, noodle3) = 3 THEN newdlesum = SUM(noodle1, noodle2, noodle3);
```

Newdlesum will be identical for all observations to noodlesum shown in Figure 2. The N function counts the number of non-missing values in the argument list. Of course, with only three variables in the list to be added, the syntax used for noodlesum is the simplest.

Other functions in this category, such as MEAN, MEDIAN, MIN, MAX, GEOMEAN, HARMMEAN, LARGEST, and SMALLEST, also ignore missing values among the arguments (unless all are missing). But for these others, this is NOT the same as treating the missing values as if they were 0's. For example, the MEAN function (which computes the arithmetic mean of the nonmissing arguments) doesn't add a zero in if there is a missing value; so, the following statements don't give the same answers if there is any missing data, as a naïve programmer might expect.

```plaintext
doodlemean1 = (doodle1 + doodle2 + doodle3 + doodle4)/4;
doodlemean2 = MEAN(doodle1,doodle2, doodle3,doodle4) ;
doodlemean3 = SUM(doodle1,doodle2,doodle3,doodle4)/4 ;
doodlemean4 = SUM(doodle1,doodle2,doodle3,doodle4)/N(doodle1,doodle2,doodle3,doodle4);
```

As shown in Figure 3, observations A and B have identical values for doodle1 – doodle4 except that in A, doodle3 is 0 and in B, it is missing. Clearly 0 and missing are not being treated the same. It is not that there is a “right” or “wrong” way that the functions should work, but it is important to understand how they behave with different inputs. And only you as the programmer can determine what is the desired way to handle missing data,
LEARNING HOW NOT TO SUBTRACT

You may be wondering why I haven’t been using the shorthand method of listing variables that have names that begin with the same prefix (e.g. referring to “doodle1 – doodle4” instead of listing “doodle1, doodle2, doodle3, doodle4”). Especially if there are a lot of variables in the list, the former method requires a lot less typing. Well, these shorthand lists can be used as arguments to functions as well, but I’ve gotten burned on this more than once. Take the two statements:

\[
\text{sumnoodle1} = \text{SUM(noodle1, noodle2, noodle3)} ;
\]
\[
\text{sumnoodle2} = \text{SUM(noodle1 - noodle3)} ;
\]

Now, of course, in a KEEP statement or an ARRAY statement, “noodle1 – noodle3” means exactly the same thing as “noodle1, noodle2, noodle3”, but this is not the case when used as the argument to one of these statistical functions. I was very surprised to see that what I thought was the sum of three positive integers could be negative (See Figure 4).

![Figure 4. Handling of variable lists in SUMs](image)

<table>
<thead>
<tr>
<th>id</th>
<th>noodle1</th>
<th>noodle2</th>
<th>noodle3</th>
<th>sumnoodle1</th>
<th>sumnoodle2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td>20</td>
<td>-6</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>-9</td>
</tr>
<tr>
<td>E</td>
<td>.</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>.</td>
</tr>
<tr>
<td>F</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>21</td>
<td>-2</td>
</tr>
<tr>
<td>H</td>
<td>6</td>
<td>2</td>
<td>.</td>
<td>8</td>
<td>.</td>
</tr>
<tr>
<td>I</td>
<td>8</td>
<td>.</td>
<td>6</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>J</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

What’s happening? The SUM function is interpreting the argument “noodle1 – noodle3” not as “noodle1” to “noodle3” (i.e. noodle1, noodle2, noodle3) but as “noodle1 MINUS noodle3”, which, of course, explains the negatives. It also explains why it is missing if either noodle1 or noodle3 is missing as there is really only a single argument to the function in the sumnoodle2 statement above, the difference between noodle1 and noodle3, which will clearly be missing if either of the components are missing. I have to confess that the first time I made this mistake, I was really flabbergasted as to why I wasn’t getting the correct sums when specifying the variable list as in sumnoodle2. Of course, there it is in the manual: “Tip: The argument list can consist of a variable list, which is preceded by OF”, but this seems so innocuous without a concrete example. The fix is simple:

\[
\text{sumnoodle3} = \text{SUM(OF noodle1 - noodle3)} ;
\]

This gives the same values as sumnoodle1 above, probably what is desired. I also learned recently (thanks to Marje Fecht, see references) that if you want to sum all the variables on a data set that start with a certain set of characters (e.g. “noodle”), you can use the following shorthand:

\[
\text{sumnoodle3} = \text{SUM(OF noodle:) ;}
\]

The above would also work if you only had noodle1, noodle3, noodle5,... (i.e. there were gaps in the numbering – which would cause an error with the noodle1-noodleN syntax) or if they didn’t have numbers at all (e.g. noodle_a, noodle_b, noodle_frenchtoast, noodle_eggplant,...). Also, you can string these “OF” variable lists together:

\[
\text{sumnoodledoodle} = \text{SUM(OF noodle1-noodle3, OF doodle1-doodle3)} ;
\]

What about the other type of variable list, the one of the form “apple - - zebra” which one expects to expand to indicate a listing of all the variables from apple to zebra in position on the data set? Will the following statement sum all the nonmissing values for an observation of the OODLES dataset (Figure 1)?:

\[
\text{bigsum1} = \text{SUM(doodle1 - strudel2)} ;
\]

In other words, will it give the same result as this statement?:

\[
\text{bigsum2} = \text{SUM(OF doodle1-doodle4, OF noodle1-noodle3, OF strudel1-strudel2)} ;
\]

No! The statement for bigsum1 is interpreted as meaning compute the sum of “noodle minus the reciprocal of strudel” or “noodle1 minus minus strudel”, or, of course, since “minus minus” equals “plus” it means sum “noodle1 plus strudel1”. The SUM function is really redundant here; again the statement for bigsum1 really has just one argument – the sum of noodle1 and noodle2. So, the result will be missing if either doodle1 or strudel2 is missing. The results are in Figure 5.
Figure 5. Double-dashed variable lists and SUMs

<table>
<thead>
<tr>
<th>id</th>
<th>doodle1</th>
<th>doodle2</th>
<th>doodle3</th>
<th>doodle4</th>
<th>noodle1</th>
<th>noodle2</th>
<th>noodle3</th>
<th>strudel1</th>
<th>strudel2</th>
<th>bigsum1</th>
<th>bigsum2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>60</td>
<td>0</td>
<td>70</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>31</td>
<td>182</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>60</td>
<td></td>
<td>70</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>179</td>
</tr>
<tr>
<td>C</td>
<td>70</td>
<td>.</td>
<td>10</td>
<td>90</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>71</td>
<td>180</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>70</td>
<td>30</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>161</td>
</tr>
<tr>
<td>E</td>
<td>70</td>
<td>80</td>
<td>30</td>
<td>60</td>
<td>.</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>70</td>
<td>252</td>
</tr>
<tr>
<td>F</td>
<td>30</td>
<td>90</td>
<td>40</td>
<td>60</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>241</td>
</tr>
<tr>
<td>G</td>
<td>90</td>
<td>80</td>
<td>90</td>
<td>10</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>90</td>
<td>292</td>
</tr>
<tr>
<td>H</td>
<td>.</td>
<td>80</td>
<td>20</td>
<td>.</td>
<td>6</td>
<td>2</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>110</td>
</tr>
<tr>
<td>J</td>
<td>70</td>
<td>10</td>
<td>50</td>
<td>.</td>
<td>8</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>70</td>
<td>145</td>
</tr>
<tr>
<td>J</td>
<td>30</td>
<td>20</td>
<td>50</td>
<td>80</td>
<td>.</td>
<td>.</td>
<td>0</td>
<td>1</td>
<td>31</td>
<td>181</td>
<td></td>
</tr>
</tbody>
</table>

Now you don’t have to use the `bigsum2` strategy. “OF” will also work with this double-dashed variable list, so `bigsum3` is the same as `bigsum2`:

```
bigsum3 = SUM(OF doodle1 -- strudel2);
```

CONCLUSIONS

There are LOTS of SAS functions and they can be extremely useful, however, as with any programming statement it is important to understand the subtleties so you get the results you intend. I hope that my explanations of some lessons I’ve learned about a few arithmetic functions will save you some headaches. These can be tricky because they are often not the kind of mistakes that generate an outright error – making them harder to detect and correct. See below for some excellent references on these – and many, many other SAS functions!

REFERENCES


ACKNOWLEDGMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.

CONTACT INFORMATION

I welcome comments, suggestions and questions at:

Christianna S. Williams, PhD
Cecil G. Sheps Center for Health Services Research
University of North Carolina at Chapel Hill
725 Martin Luther King, Jr. Boulevard
Campus Box # 7590
Chapel Hill, North Carolina 27599
Email: Christianna_Williams@unc.edu