ABSTRACT
There probably are not 50 ways to lose your data, but there are a bunch. Of course, if we all backed up our data before working with it, there wouldn’t be a problem. However, that’s not always a practical solution, especially if your data sets are large. By being aware of coding syntax that can lead to data destruction, you can avoid finding yourself with a data set containing 0 observations. The techniques in this paper can help anyone from the novice to the experienced user prevent disasters. If you’ve never had this problem, these techniques will allow you to continue to be quite proud of yourself.

INTRODUCTION
As a programmer, you know how important it is to review your program before you submit it, especially if you are manipulating an existing data set. In addition to destroying an entire data set, there are many logic errors, syntax errors, and even human errors that destroy data integrity. This paper groups possible destructive techniques and their solutions into the following categories:

- Novice problems
- Not knowing your data
- Syntax errors
- Bad logic
- Intentionally deleting data sets
- Hardware problems

NOVICE PROBLEMS
Many errors that beginning SAS® programmers encounter fall into other categories; however, three are identified as common misconceptions.

1. Assuming data must be created before it can be used.

```sas
data perm.DataSet;
proc print data = perm.DataSet;
run;
```

**Solution:** Refer to the permanent data set in the PRINT procedure. It is not necessary to have a DATA step in order to use a data set name in the PROC statement.

```sas
proc print data = perm.DataSet;
run;
```

2. Not keeping all of the variables that you want to keep.

```sas
data NewData;
   set OldData; /*The data set OldData contains the variables X, Y, and Z. */
   keep Z; /*The only variables kept should be the variables W and Z. */
   W = X + Y;
run;
```

**Solution:** Specify all of the variables you want to keep (even new ones that are created in the DATA step) in the KEEP statement.

```sas
data NewData;
   set OldData;
   keep Z W;
   W = X + Y;
run;
```
3. Not knowing the names of your permanent data sets and creating a new one with the name of an existing data set.

data perm.old_data;
   set other_data;
   if x = . then x = 5;
run;

   Solution: Look at the names of the existing permanent data sets before creating a new one.

   proc datasets lib = perm nods;
      contents data = _all_; 
   run;

data perm.new_name;
   set other_data;
   if x = . then x = 5;
run;

NOT KNOWING YOUR DATA

The first rule of programming should be 'know thy data.' Unfortunately, many programmers rush into their programming tasks without considering the values of their data. The following are examples of ignoring that rule.

4. Not knowing the case of character data.

   data you;
      set perm.Names;
      where FirstName contains 'il';
   run;

   Solution: Use the UPCASE function when comparing character variables with constants.

   data you;
      set perm.Names;
      where UPCASE(FirstName) contains 'IL';
   run;

5. Incorrectly choosing the length of numeric data.

   data five;
      length num5 5 num8 8;
      do num8 = 10000000000 to 100000000000 by 100000000000;
      num5 = num8;
      output;
      end;
run;

   Here are the observations six through eight in the data set FIVE. Notice that the values of NUM5 and NUM8 are not equal starting with observation seven, even though the assignment statement NUM5=NUM8 makes them equal.

<table>
<thead>
<tr>
<th>Obs</th>
<th>num5</th>
<th>num8</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>510000000000</td>
<td>510000000000</td>
</tr>
<tr>
<td>7</td>
<td>609999998976</td>
<td>610000000000</td>
</tr>
<tr>
<td>8</td>
<td>709999998976</td>
<td>710000000000</td>
</tr>
</tbody>
</table>
Solution: Do not change the length of numeric data if the values are large numbers.

data five;
  do num8 = 10000000000 to 10000000000000 by 100000000000;
    num5 = num8;
  output;
end;
runk;

6. Incorrectly determining the length of character data.

   trainer.dat

   Linda Jolley, Technical Trainer
   Jane Stroupe, Technical Trainer

   data instructor;
     infile 'trainer.dat' dlm = ',';
     input Name $ Job;
   run;

   The resulting data follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linda Jo</td>
<td>Technica</td>
</tr>
<tr>
<td>Jane Str</td>
<td>Technica</td>
</tr>
</tbody>
</table>

   Solution: When using list input, specify the length of the character data. List input is used to read this data since it is free format data. The default length of character data is 8, which can sometimes result in data truncation.

   data instructor;
     length Name $12 Job $17;
     infile 'trainer.dat' dlm = ',';
     input Name $12. Job $17.;
   run;

   Alternate Solution: Use the colon modifier to specify an informat in the INPUT statement.

   data instructor;
     infile datalines dlm = ',';
     input Name :$12. Job :$17.;
   run;

7. Having large numeric data that results in precision problems.

   data BigNumber_Num;
     input Num;
datalines;
12345678901234567890
;run;

   Numeric data can store 16 or 17 significant digits in a length of 8; therefore, if a numeric value has more than 16 or 17 digits, you lose precision. The resulting value of NUM in this example is 12345678901234567168.

   Solution: Create a character value instead of a numeric value. The value of this character variable would be 12345678901234567890.

   data BigNumber_Char;
     input Num :$20.;
datalines;
12345678901234567890
;run;
8. Creating a numeric variable from a character variable.

```sas
data Try_Again;
  set BigNumber_Char;
  NewNum = input(Num, 20.);
  format NewNum 20.;
run;
```

**Solution:** The problem continues to be the length of the original number. Even if you create a character value instead of a numeric one for the variable NUM and then attempt to use the INPUT function to create a numeric variable from the character value, you lose precision.

9. Using PROC IMPORT to determine the type of variable.

```sas
proc import out = work.teens datafile = "teens.xls" dbms = excel replace;
run;
```

The Excel spreadsheet:

<table>
<thead>
<tr>
<th>ID</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>623</td>
<td>16</td>
</tr>
<tr>
<td>321</td>
<td>13</td>
</tr>
<tr>
<td>A145</td>
<td>14</td>
</tr>
<tr>
<td>4632</td>
<td>14</td>
</tr>
<tr>
<td>C2314</td>
<td>15</td>
</tr>
<tr>
<td>1245</td>
<td>16</td>
</tr>
</tbody>
</table>

**Solution:** Use the MIXED=YES option in the PROC IMPORT statement. The MIXED=YES option assigns a SAS character type for the column and converts all numeric data values to character data values when mixed data types are found.

```sas
proc import out = work.teens datafile = "teens.xls"
  dbms = excel replace;
  mixed=yes;
run;
```


```sas
data LongRecl;
  infile 'LongRec.dat';
  input Char $480.;
  Len = length(Char);
run;
```

**Solution:** Use the LRECL= option in the INFILE statement since the record length is longer than the default (256 bytes in Windows and UNIX).

```sas
data LongRecl;
  infile 'LongRec.dat' lrecl = 480;
  input Char $480.;
  Len = length(Char);
run;
```

11. Not knowing that the raw data has varying lengths.

```sas
numbers.dat
```

```
143
36921
2
4509
```


```sas
data varying;
  infile 'numbers.dat';
  input Num 6.;
run;
```

The result of the DATA step follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36921</td>
</tr>
<tr>
<td>2</td>
<td>4509</td>
</tr>
</tbody>
</table>

**Solution:** See Number 12.

12. Not knowing that the raw data has varying lengths (Version 2).

```sas
data varying;
  infile 'numbers.dat' missover;
  input Num 6.;
run;
```

The result of the DATA step follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
</tr>
</tbody>
</table>

**Solution to both:** Use the TRUNCOVER option in the INFILE statement.

```sas
data varying;
  infile 'numbers.dat' truncover;
  input Num 6.;
run;
```

The result of the DATA step follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>143</td>
</tr>
<tr>
<td>2</td>
<td>36921</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4509</td>
</tr>
</tbody>
</table>

13. Having multiple delimiters treated as one delimiter.

```sas
data multiple;
  infile 'fullnames.dat' dlm = ', ';
  input LastName : $8. FirstMName : $25.;
run;
```

```
Thomas, John Marshall
Sherrill, Franklin Woodford
```

**Foundations and Fundamentals**

**SAS Global Forum 2009**
The result of this DATA step follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>LastName</th>
<th>MName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thomas</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Sherrill</td>
<td>Franklin</td>
</tr>
</tbody>
</table>

**Solution:** Use the DLMSTR= option instead of the DLM= option in the INFILE statement. This SAS 9.2 option specifies that the delimiter must be the exact string. In this example, the delimiter must be a comma followed by a blank.

```sas
data multiple;
  infile 'fullnames.dat' dlmstr = ', ';  
  input LastName : $8. FirstMName : $25.;
run;
```

The result of this DATA step follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>LastName</th>
<th>FirstMName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thomas</td>
<td>John Marshall</td>
</tr>
<tr>
<td>2</td>
<td>Sherrill</td>
<td>Franklin Woodford</td>
</tr>
</tbody>
</table>

14. Having class variables that are missing when creating statistics on an analysis variable.

```sas
proc summary data = perm.class nway;
  output out = summary mean = AvgAge;
  class Group;
  var Age;
run;
```

For most procedures that calculate statistics, missing CLASS variables are not included in the output data sets. The data used for this example is

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Age</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michelle</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Steve</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>James</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>Jeffrey</td>
<td>13</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>John</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Louise</td>
<td>12</td>
<td>D</td>
</tr>
<tr>
<td>21</td>
<td>Mary</td>
<td>15</td>
<td>D</td>
</tr>
</tbody>
</table>

The resulting data created by the SUMMARY procedure is as follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Group</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>AvgAge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>1</td>
<td>4</td>
<td>12.0000</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>1</td>
<td>6</td>
<td>14.3333</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>1</td>
<td>1</td>
<td>11.0000</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>1</td>
<td>8</td>
<td>13.5000</td>
</tr>
</tbody>
</table>
Solution: Use the MISSING option in the PROC SUMMARY statement.

```sas
proc summary data = perm.class missing nway;
  output out = summary mean = AvgAge;
  class Group;
  var Age;
run;
```

The resulting data created by this SUMMARY procedure is as follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Group</th>
<th><em>TYPE</em></th>
<th>FREQ_</th>
<th>AvgAge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>15.0000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1</td>
<td>4</td>
<td>12.0000</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>1</td>
<td>6</td>
<td>14.3333</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>1</td>
<td>1</td>
<td>11.0000</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>1</td>
<td>8</td>
<td>13.5000</td>
</tr>
</tbody>
</table>

SYNTAX ERRORS

Often syntax errors stop execution of a step; therefore, a DATA step might not remove data from your data set. However, there are some syntax errors that have a destructive behavior. Watch out for them!

15. Using the SQL procedure with too many semicolons.

```sas
proc sql;
  delete from test;
  where name='John';
quit;
```

Although this PROC SQL step causes a syntax error, it is too late. All of the rows of data have been deleted.

**Solution:** Put a semicolon only at the end of the SQL DELETE statement.

```sas
proc sql;
  delete from test
  where name='John';
quit;
```

**Alternate Solution:** Use the NOEXEC option in the PROC SQL statement. This option checks the syntax of a PROC SQL step without actually executing it.

```sas
proc sql noexec;
  delete from test
  where name='John';
quit;
```

16. Omitting a semicolon and misspelling a keyword.

```sas
data two;
  ser perm.one;
run;
```

This program creates three data sets: TWO, SER, and PERM.ONE. Since there are no other statements in the DATA step, the data set PERM.ONE is created with no observations.

**Solution:** Add a semicolon at the end of the DATA statement and/or spell SET correctly. (The default SAS system option DATASTMTCHK = COREKEYWORDS prohibits the SET keyword as a one-level SAS data set name.)

```sas
data two;
  set perm.one;
run;
```

OR
data two
set perm.one;
run;

17. Hitting the incorrect key on the keyboard causing more misspellings.

   proc sql;
   delect * from perm.one;
   quit;

   Log

   proc sql;
   delect * from one;
   ------
   1
   WARNING 1-322: Assuming the symbol DELETE was misspelled as delect.
   NOTE: 3 rows were deleted from WORK.ONE.
   quit;

**Solution:** Watch your fingers as you type. SQL interprets DELECT as DELETE and deletes all the rows of data from the data set PERM.ONE. The 'S' and the 'D' are next to each other on a standard keyboard.

   proc sql;
   select * from one;
   quit;

18. Merging without a BY statement.

   data set CHAR1    data set CHAR2
   one    two
   A      B
   C      D
   E      F

   data char3;
   merge char1 char2;
   run;

   Without a BY statement, the merge combines the first observations of each data set, the second observations, and so forth. Therefore, there is incorrect data, and much of it is lost. This is the resulting data.

<table>
<thead>
<tr>
<th>Obs</th>
<th>one</th>
<th>two</th>
<th>three</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>D</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>F</td>
<td>Z</td>
</tr>
</tbody>
</table>

**Solution:** Use a BY statement.

   data char3;
   merge char1 char2;
   by a; /* the input data is already sorted */
   run;
The resulting data set now contains the correct data:

<table>
<thead>
<tr>
<th>Obs</th>
<th>one</th>
<th>two</th>
<th>three</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>D</td>
<td>Z</td>
</tr>
</tbody>
</table>

19. Merging on one common variable when there is more than one common variable.

-data set CHAR1
-one
two
-A
-C
-E
-data set CHAR4
-one
two
-A
-B
-C

-data char5;
  merge char1 char4;
  by one; /* the data is already sorted */
-run;

When the two data sets are merged only on the variable A, the values of B from the data set CHAR4 overwrite the values of B from the data set CHAR1. The resulting data set follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>one</th>
<th>two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Z</td>
</tr>
</tbody>
</table>

Solution: Merge on all like-named variables.

-data char6;
  merge char1 char4;
  by one two; /* the data is already sorted */
-run;

The resulting data follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>one</th>
<th>two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>Z</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Alternate Solution: Rename all of the like-named variables except the variable on which the merge should be done.

-data char7;
  merge char1 char4(rename=(Two=NewTwo));
  by one; /* the data is already sorted */
-run;
The resulting data follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>one</th>
<th>two</th>
<th>New Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>D</td>
<td>Z</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

20. Forgetting to reset an option.

```bash
options obs = 10;

proc print data = perm.one;
run;
proc sort data = perm.one;
    by x;
run;
```

Solution: Reset the option. In the case of the OBS = option, use OBS = MAX to read all of the data.

```bash
options obs = max;
proc sort data = perm.one;
    by x;
run;
```

21. Reading in delimited data incorrectly.

```bash
data locations;
    infile 'locations.dat';
    input City : $12. County : $10. State : $2.;
run;
```

The problem here is that no delimiter has been specified; therefore, the default delimiter of a blank is being used. The resulting data set follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>City</th>
<th>County</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Libertyville</td>
<td>Lake, IL</td>
<td>IL</td>
</tr>
</tbody>
</table>

Solution: See Number 23.

22. Reading in delimited data incorrectly (Version 2).

```bash
data locations;
    infile 'locations.dat' dlm = ',';
    input City : $12. County : $10. State : $2.;
run;
```

This is a good start to fixing the problem. However, multiple delimiters are treated as one delimiter, causing the INPUT statement to reach the end of the line in the second observation. The resulting data set is no different than it was for the previous example.

Solution: See Number 23.
23. Reading in delimited data incorrectly (Version 3).

```sas
data locations;
    infile 'locations.dat' dlm = ',' missover;
    input City : $12. County : $10. State : $2.;
run;
```

The MISOVER option prevents the INPUT statement from reaching the end of the line. However, for the second observation, the value of COUNTY is read in as KS. The resulting data follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>City</th>
<th>County</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Libertyville</td>
<td>Lake</td>
<td>IL</td>
</tr>
<tr>
<td>2</td>
<td>Kansas City</td>
<td></td>
<td>KS</td>
</tr>
</tbody>
</table>

**Solution to all three:** Use the DSD option in the INFILE statement.

```sas
data locations;
    infile 'locations.dat' dlm = ',' missover dsd;
    input City : $12. County : $10. State : $2.;
run;
```

The DSD option (Delimiter-Sensitive Data) uses a comma as the default delimiter, treats consecutive commas as a missing value, and reads a character value enclosed in quotation marks and removes the quotation marks from the character string before the value is stored. The resulting data set follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>City</th>
<th>County</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Libertyville</td>
<td>Lake</td>
<td>IL</td>
</tr>
<tr>
<td>2</td>
<td>Kansas City</td>
<td></td>
<td>KS</td>
</tr>
</tbody>
</table>

24. Reading mixed record types incorrectly.

```sas
data Holiday;
    infile 'holiday.dat' dsd;
    input Country : $2. Holiday : $16.;
    if Country = 'US' then input Date : date9.;
    else input Date : ddmmyy10.;
    format date mmddyy10.;
run;
```

The resulting data follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Country</th>
<th>Holiday</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>US</td>
<td>Independence Day</td>
<td>.</td>
</tr>
</tbody>
</table>

**Solution:** Use multiple INPUT statements with a single trailing @ sign and check a condition before reading more data. The single trailing @ sign holds the current record in the input buffer so that the second INPUT statement can continue to read from that record.

```sas
data Holiday;
    infile 'holiday.dat' dsd;
    input Country : $2. Holiday : $16. @;
    if Country = 'US' then input Date : date9.;
    else input Date : ddmmyy10.;
    format date mmddyy10.;
run;
```
Alternate Solution: Use the DATASTYLE= SAS system option and the ANYDTDTE informat on the Date variable. The DATASTYLE= SAS system option identifies the order of month, day, and year. The default value of LOCALE uses the style that has been specified by the LOCALE= system option. The ANYDTDTE informat reads dates in various forms.

```sas
options datestyle = locale;
data Holiday;
  infile 'holiday.dat' dsd;
  format date mmddyy10.;
run;
```

25. Using the NODUPKEY option with PROC SORT and losing the duplicate observations.

```sas
data test;
  infile datalines @@;
  input Name $;
  datalines;
  John Bill Sam Bill Tom Joe Chris Bill John Kent Chris
  ;
  run;

proc sort data = test nodupkey;
  by Name;
run;
```

Solution: Use the DUPOUT= option in the PROC SORT statement and name a data set that contains the duplicate observations.

```sas
proc sort data = test nodupkey dupout = extras;
run;
```

26. Merging to make changes to data.

```sas
data set Grocery data set Update

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1065</td>
<td>Bob</td>
<td>Clerk</td>
<td>35090</td>
</tr>
<tr>
<td>1112</td>
<td>Sue</td>
<td>Manager</td>
<td>55890</td>
</tr>
<tr>
<td>1117</td>
<td>Sam</td>
<td>Bagger</td>
<td>31298</td>
</tr>
<tr>
<td>1350</td>
<td>Peg</td>
<td>Stocker</td>
<td>25897</td>
</tr>
<tr>
<td></td>
<td>John</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bill</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sam</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bill</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Joe</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chris</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bill</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>John</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kent</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chris</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```sas
data updated_grocery;
  merge grocery update;
  by idnum;
run;
```

Because the MERGE processes data sequentially and the like-named variables in the data set Update overwrite the variables from Grocery, the resulting data set is improperly updated. The resulting data set follows.

```sas
Obs    ID        Name      Job Salary
1      1065                Manager     45125
2      1112      Sue       Manager     55890
3      1117                            33589
4      1350                Bagger
5      1350                            30876
```
Solution: Use the UPDATE statement instead of the MERGE statement.

```sas
data updated_grocery;
  update grocery update;
  by idnum;
run;
```

The UPDATE statement changes the values of selected observations in a master file by applying transactions. If the data in the transaction data set is missing, the UPDATE statement does not overwrite existing values in the master data set with missing ones in the transaction data set. The resulting data set follows.

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1065</td>
<td>Bob</td>
<td>Manager</td>
<td>45125</td>
</tr>
<tr>
<td>2</td>
<td>1112</td>
<td>Sue</td>
<td>Manager</td>
<td>55890</td>
</tr>
<tr>
<td>3</td>
<td>1117</td>
<td>Sam</td>
<td>Bagger</td>
<td>33589</td>
</tr>
<tr>
<td>4</td>
<td>1350</td>
<td>Peg</td>
<td>Bagger</td>
<td>30876</td>
</tr>
</tbody>
</table>

27. Reading hierarchical files and losing the last observation.

```sas
data pets;
  retain Name;
  keep Name NumDogs NumCats NumOther;
  infile 'pets.dat';
  input @1 FirstChar : $1. @;
  if FirstChar in ('1','2','3','4','5','6','7','8','9') then do;
    if _n_ ne 1 then output;
    input Name : $8.;
    NumDogs = 0;
    NumCats = 0;
    NumOther = 0;
  end;
  else do;
    input @1 PetName : $15. Pet : $3.;
    if upcase(Pet) = 'DOG' then NumDogs + 1;
    else if upcase(Pet) = 'CAT' then NumCats + 1;
    else NumOther + 1;
  end;
run;
```

The resulting data set follows.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Num Dogs</th>
<th>Num Cats</th>
<th>Num Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michelle</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Theresa</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Solution: Specify the END= option in the INFILE statement and use conditional logic to output the last observation.

data pets;
    retain Name;
    keep Name NumDogs NumCats NumOther;
    infile 'pets.dat' end = LastObs;
    input @1 FirstChar : $1. @;
    if FirstChar in ('1','2','3','4','5','6','7','8','9') then do;
        if _n_ ne 1 then output;
        input Name : $8.;
        NumDogs = 0;
        NumCats = 0;
        NumOther = 0;
    end;
    else do;
        input @1 PetName : $15. Pet : $3.;
        if upcase(Pet) = 'DOG' then NumDogs + 1;
        else if upcase(Pet) = 'CAT' then NumCats + 1;
        else NumOther + 1;
    end;
    if LastObs then output;
run;

The resulting data set follows.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>NumDogs</th>
<th>NumCats</th>
<th>NumOther</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michelle</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Theresa</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Nancy</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

28. Allowing automatic conversion of numeric to character values.

Data set Numbers

<table>
<thead>
<tr>
<th>NumValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
</tr>
<tr>
<td>5498</td>
</tr>
<tr>
<td>5698</td>
</tr>
</tbody>
</table>

data character;
set Numbers;
CharValue = substr(NumValue, 1, 4);
run;

When automatic conversion of numeric data to character values occurs, the digits are written out in 12 spaces and are right aligned; therefore, the first four characters would be blank. The resulting data set follows.

<table>
<thead>
<tr>
<th>Obs</th>
<th>NumValue</th>
<th>CharValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5498</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5698</td>
<td></td>
</tr>
</tbody>
</table>
Alternate Solution: Use the PUT function with a specific number of digits if you know the number of digits.

```sas
data character;
  set Numbers;
  CharValue = substr(put(NumValue, 4.), 1, 4);
run;
```

Alternate Solution: Use LEFT function along with the PUT function with a specific number of digits if you do not know the number of digits

```sas
data character;
  set Numbers;
  CharValue = substr(left(put(NumValue, 4.)), 1, 4);
run;
```

29. Not specifying the KEY variable as a DATA item when using the hash object with the OUTPUT method.

```sas
data _null_; length List $100;
  if _n_ = 1 then do;
    declare hash ph();
    ph.DefineKey('Date');
    ph.DefineData('List');
    ph.DefineDone();
  end;
  set PhoneCalls end = last;
  if ph.find() ne 0 then do;
    List = PhoneNum;
    ph.add();
  end;
  else do;
    List = catx(', ', List, PhoneNum);
    ph.replace();
  end;
  if last then ph.output(dataset:'PhoneList');
run;
```

A partial listing of the data set PhoneCalls follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Date</th>
<th>PhoneNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21DEC2006</td>
<td>324-6674</td>
</tr>
<tr>
<td>2</td>
<td>21DEC2006</td>
<td>312-5098</td>
</tr>
<tr>
<td>3</td>
<td>21DEC2006</td>
<td>312-9088</td>
</tr>
<tr>
<td>4</td>
<td>21DEC2006</td>
<td>324-3452</td>
</tr>
<tr>
<td>5</td>
<td>21DEC2006</td>
<td>312-7483</td>
</tr>
<tr>
<td>6</td>
<td>21DEC2006</td>
<td>324-8943</td>
</tr>
<tr>
<td>7</td>
<td>21DEC2006</td>
<td>312-7456</td>
</tr>
</tbody>
</table>

The variable DATE is not in the resulting data set.

<table>
<thead>
<tr>
<th>Obs</th>
<th>List</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>324-6674, 312-5098, 312-9088, 324-3452, 312-7483, 324-8943, 312-7456, 324-4544</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>312-5098, 312-4755, 389-0098, 324-6674</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>312-9088, 324-3452, 312-7456, 312-5098, 324-8943, 312-5098</td>
<td></td>
</tr>
</tbody>
</table>
Solution: Add the KEY variable as one of the DATA columns in the DEFINEDATA method. By default, the KEY variable is not included as part of the data.

```sas
data _null_ ;
  length List $100;
  if _n_ = 1 then do;
    declare hash ph();
    ph.DefineKey('Date');
    ph.DefineData('Date', 'List');
    ph.DefineDone();
  end;
  set PhoneCalls end = last;
  if ph.find() ne 0 then do;
    List = PhoneNum;
    ph.add();
  end;
  else do;
    List = catx(', ', List, PhoneNum);
    ph.replace();
  end;
  if last then ph.output(dataset:'PhoneList');
run;
```

30. Having duplicate keys when using a hash object. Only the first one is loaded into the hash object.

```sas
data all_inquiries;
  drop rc;
  length Product_ID 8;
  if _N_ = 1 then do;
    declare hash h(dataset:'orders');
    h.defineKey('Order_ID');
    h.defineData('Order_ID', 'Product_ID');
    h.defineDone();
    call missing(Product_ID);
  end;
  set inquiries;
  rc = h.find();
run;
```

Solution: Use the MULTIDATA : 'YES' parameter in the DECLARE statement. MULTIDATA is a SAS 9.2 enhancement to the hash object syntax which allows multiple data items for each non-unique key.

```sas
data all_inquiries;
  drop rc;
  length Product_ID 8;
  if _N_ = 1 then do;
    declare hash h(dataset:'orders', multidata:'yes');
    h.defineKey('Order_ID');
    h.defineData('Order_ID', 'Product_ID');
    h.defineDone();
    call missing(Product_ID);
  end;
  set inquiries;
  rc = h.find();
do while (rc = 0);
  output;
  rc = h.find_next();
end;
run;
```
31. Not including automatic variables in your data.

```sas
proc sort data = perm.class out = class;
   by Group;
run;

data Summary;
   set class nobs = NumObs;
   keep Group AvgAge TotAge NumObs Count;
   by Group;
   if first.Group then do;
      TotAge = 0;
      Count = 0;
   end;
   TotAge + Age;
   Count + 1;
   if last.Group then do;
      AvgAge = TotAge / Count;
      output;
   end;
run;

Solution: Create a DATA step variable that contains the value of the NOBS= variable, NUMOBS.

```sas
data Summary;
   set class nobs = NumObs;
   keep Group AvgAge TotAge TotObs Count;
   TotObs = NumObs;
   by Group;
   if first.Group then do;
      TotAge = 0;
      Count = 0;
   end;
   TotAge + Age;
   Count + 1;
   if last.Group then do;
      AvgAge = TotAge / Count;
      output;
   end;
run;
```sas

32. Reading from multiple raw data files in one DATA step.

There are three raw data files with the same fields but a different number of records. The DATA step stops when execution reaches the end of file for the raw data file with the least number of records. Therefore, the DATA step never reads the rest of the other raw data files.

```sas
data names;
   infile 'names1.dat';
   input Name $ Age;
   output;
   infile 'names2.dat';
   input Name $ Age;
   output;
   infile 'names3.dat';
   input Name $ Age;
   output;
run;
```
Solution: Use the FILEVAR = option in the INFILE statement to specify a variable whose change in value causes the INFILE statement to close the current input file and open a new one. In addition, use a DO WHILE loop to read all of the records of each raw data file.

```sas
data routes;
   length Readit $10;
   input ReadIt;
   infile in filevar = ReadIt end = LastFile pad;
   do while(LastFile = 0);
      input Name $ Age;
      output;
   end;
datalines;
names1.dat
names2.dat
names3.dat
;
run;
```

BAD LOGIC

33. Having a never met condition on a WHERE statement.

```sas
proc sort data = one;
   by x;
   where x = 12;
run;
```

Solution: Create a data set using the OUT= option. In this case, the data set TWO would be empty, but the integrity of the data set ONE is maintained.

```sas
proc sort data = one out = two;
   by x;
   where x = 12;
run;
```

34. Omitting a condition on an IF statement.

```sas
data one;
   set one;
   if x then delete;
run;
```

Solution: By not specifying a value for the variable X, the condition IF X is true as long as X is not 0 and X is not missing. Therefore, all of the observations with a nonzero value of X are deleted.

```sas
data one;
   set one;
   if x = 12 then delete;
run;
```

35. Specifying a false condition when using the DATA step with a MODIFY statement and a WHERE statement.

```sas
data one;
   modify one;
   x = x * 2;
   where x = 12;
run;
```
Solution: Use the IF statement instead of the WHERE statement.

```sas
data one;
  modify one;
  x = x * 2;
  if x = 12;
run;
```

36. Omitting a LENGTH statement.

```sas
data char;
  input a $ b $;
  datalines;
  ONE TWO
  THREE FOUR
  FIVE SIX
  SEVEN EIGHT
  NINE TEN;
  data NewChar;
    set char;
    A = A||B;
  run;
```

Solution: Use a LENGTH statement before the SET statement.

```sas
data NewChar;
  length A $ 16;
  set char;
  A = A||B;
run;
```

37. Using the SCAN function with multiple delimiters.

```sas
data test;
  input pets : $20.;
  pet2 = scan(pets, 2, ',');
  datalines;
   cat,dog,fish
   dog,,bird
  ;
run;
```

Solution: By default, the SCAN function treats multiple delimiters as one delimiter. In addition, any delimiters before or after the string are ignored. Use the 'M' option as the fourth option in the SCAN function (a 9.2 enhancement). It specifies that multiple consecutive delimiters (and any at the beginning or end of the first argument) are treated as missing values between the delimiters; therefore, the variable being created has a missing value.

```sas
data test;
  input pets : $20.;
  pet2=scan(pets, 2, ',', 'M');
  datalines;
  cat,dog,fish
  dog,,bird
  ;
run;
```
38. Creating a random sample with the RANUNI function when you need to recreate the same sample later.

```sas
data subset;
   do I = 1 to 10;
      x = int(num * ranuni(0));
      set Seuss point = x nobs = num;
      output;
   end;
   stop;
run;
```

Solution: Use a positive integer as the seed in the RANUNI function.

```sas
data subset;
   do I = 1 to 10;
      x = int(num * ranuni(12345));
      set Seuss point = x nobs = num;
      output;
   end;
   stop;
run;
```

### INTENTIONALLY DELETING DATA SETS

39. Using the DATASETS procedure to delete the entire data set.

```sas
proc datasets lib = work;
   delete one;
run;
```

40. Using the SQL procedure to drop a data set.

```sas
proc sql;
   drop table one;
quit;
```

41. Using the Explorer window to delete a data set.

   Right-click on a SAS data set in the Explorer window and select DELETE. Be careful because this does not put the data set into the Windows recycle bin.

42. Using the VIEWTABLE window to delete rows from a SAS data set.

   - Double-click on the data set in the Explorer window.
   - From the Menu bar, select EDIT → Edit Mode.
   - Click on the row to be deleted.
   - From the Menu bar, select EDIT → Delete Row.

43. Use PROC SQL with the DELETE FROM statement correctly.

```sas
proc sql;
   delete from one where x = 12;
quit;
```

### HARDWARE PROBLEMS

44. Losing your laptop; for example, leaving it at security at the airport.

45. Having your hard drive crash.

46. System interruptions; for example, IT takes the system down for maintenance while your program is executing.

47. Backing up your data on tape and then destroying the tape.

48. Power outages during program execution.
Solution to all of these: Back up your data.

CONCLUSION
There might not be 50 ways to lose your data, but there are many. We just hope you cannot think of any more, 48 are enough!

Furthermore, we hope you never accidentally lose your data because we hope you always back it up. Make sure you have no syntax errors or logic mistakes. In addition, it is critical that you know your data. Using the CONTENTS procedure and the PRINT procedure to familiarize yourself with the data attributes and the data values can help prevent data destruction.

Always remember: Think before you submit.

ACKNOWLEDGMENTS
Thanks to the following instructors who gave us examples of losing data, which of course, they never did:
Kay Alden, Michelle Buchecker, Lise Cragen, Davetta Dunlap, David Ghan, Amy Gumm, Sue Rakes, Kent Reeve, Don Reid, Jim Simon

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:

Linda Jolley, Technical Training Specialist
SAS Institute Inc.
Kansas City Regional Office
Phone: (913) 491-1166
E-mail: linda.jolley@sas.com

Jane Stroupe, Technical Training Specialist
SAS Institute Inc.
Chicago Regional Office
Phone: (847) 367-7216
E-mail: jane.stroupe@sas.com

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.