# SAS Programming (BIOL-4V190)

## Chapter 8

## Performing Iterative Processing: Looping

## 8.1 Introduction

Looping is used when a code segment must be run more than one time.
This is accomplished through the use of DO, DO WHILE, and DO UNTIL statements.

## 8.2 DO Groups

A DO group is created by placing programming statements within a set of DO-END statements.
These statements are only executed if the condition on the DO statement is met.
DO groups can be used to replace multiple IF-THEN statements and like the SELECT statement, usually creates code that is easier to understand and maintain.

The example in this section also illustrates the use of the DELETE statement.
The DELETE statement causes the current observation to not be written to the data set.

We will look at two examples that generate the same results, one using IF-THEN statements and the other using a do loop.

Program 8-1 uses a series of IF-THEN statements to create a new categorical variable, Agegrp, and to calculate a new variable, Grade.

```
if Age le 39 then Agegrp = 'Younger group';
if Age le 39 then Grade  = .4*Midterm + .6*FinalExam;
if Age gt 39 then Agegrp = 'Older group';
if Age gt 39 then Grade  = (Midterm + FinalExam)/2;
```

Program 8-2 uses a do loop to accomplish the same result.

In this example, if age is less than or equal to 39, then the statements in the first do loop are executed, and if Age is greater than 39, then the statements in the second do loop are executed.

Notice that the syntax requires an END statement at the bottom of each DO loop.

```
if Age le 39 then do;
  Agegrp = 'Younger group';
  Grade = .4*Midterm + .6*FinalExam;
end;
else if Age gt 39 then do;
  Agegrp = 'Older group';
  Grade = (Midterm + FinalExam)/2;
end;
```

In both programs, observations with missing ages are deleted because of this statement:

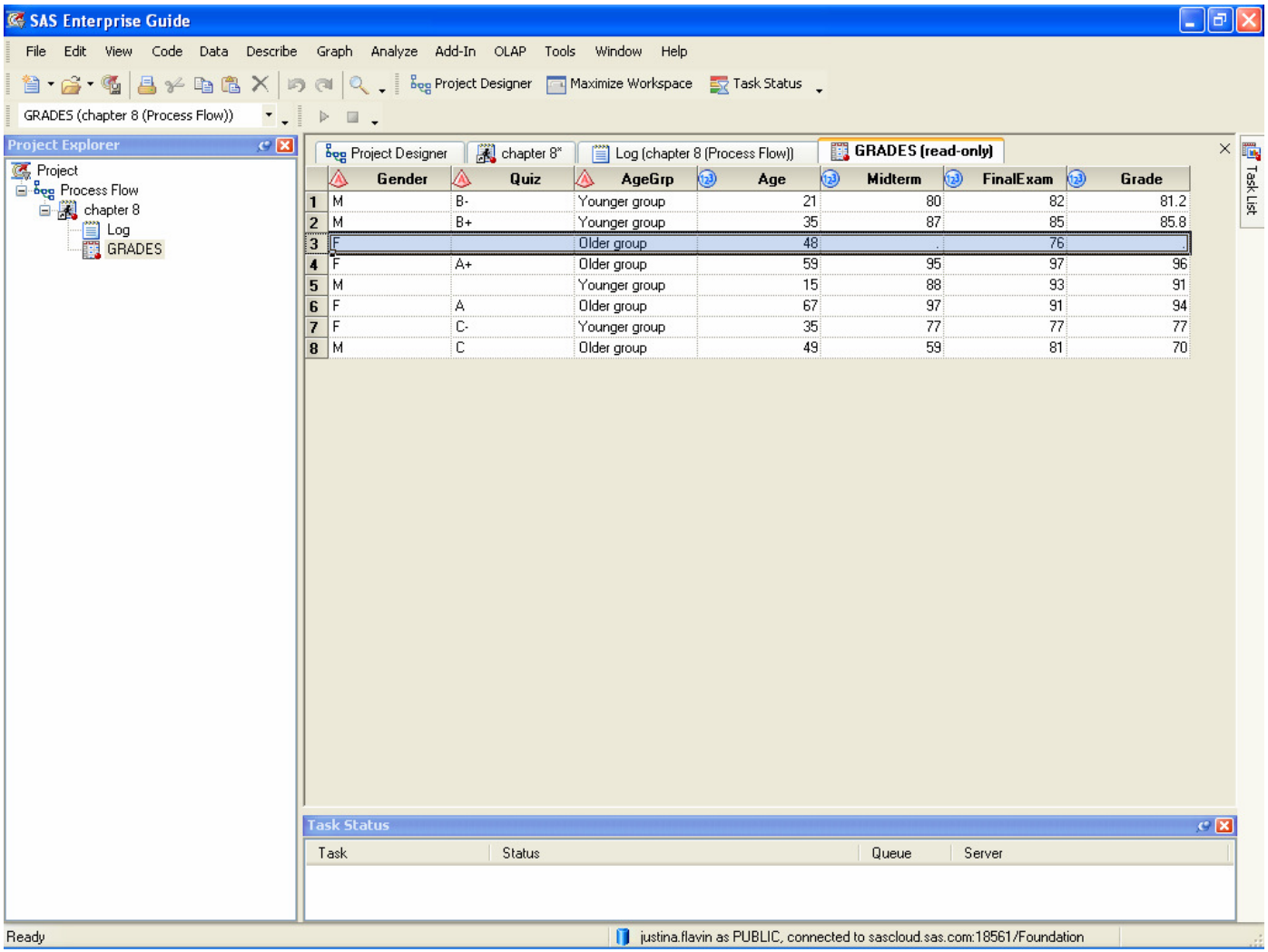
```
if missing(Age) then delete;
```

So while SAS reads all data lines into the PDV, only those having a non-missing value for Age are written to the data set.

Had you wished to keep the records with missing Age, the code could be modified to prevent execution of the first two if statements and the first do loop as follows:

```
if . < Age le 39 then Agegrp = 'Younger group';
if . < Age le 39 then Grade  = .4*Midterm + .6*FinalExam
if . < Age le 39 then do;
```

Here is the data set GRADES produced by Program 8-2




| | Gender | Quiz | AgeGrp | Age | Midterm | FinalExam | Grade |
|---|---|---|---|---|---|---|---|
| 1 | M | B- | Younger group | 21 | 80 | 82 | 81.2 |
| 2 | M | B+ | Younger group | 35 | 87 | 85 | 85.8 |
| 3 | F | | Older group | 48 | . | 76 | . |
| 4 | F | A+ | Older group | 59 | 95 | 97 | 96 |
| 5 | M | | Younger group | 15 | 88 | 93 | 91 |
| 6 | F | A | Older group | 67 | 97 | 91 | 94 |
| 7 | F | C- | Younger group | 35 | 77 | 77 | 77 |
| 8 | M | C | Older group | 49 | 59 | 81 | 70 |

Another interesting thing to look at in the data set is the value of Grade in the observation that has a missing value for Midterm.

Grade is calculated using Midterm and FinalExam.

When calculating a new variable using an assignment statement, if there is a missing value in one or more variables used in the calculation, then the new variable is set to missing.

This is important to remember since there is another way to perform the calculation.

Using a function, SAS will ignore the missing values and perform the calculation with whatever variables that are available – we will learn how to do this in a later lecture.

Which way you perform the calculation is very much dependent on how you wish to handle the missing values.

In this example, we are calculating a final grade.

If the student did not show up to take the Midterm, then the missing value for Midterm should be ignored and her value of Grade should probably be equal to FinalExam/2.

However, if she was ill, the instructor may allow her to make up the missed Midterm, so assignment of a final grade is not appropriate at this time.

You probably noticed the use of the length statement in Programs 8-1 and 8-2.
The length statement allows you to declare or set up variables before assigning values to them.
In these examples, the length statement could be removed if the input statement was changed to:

```
input Age Gender $ Midterm Quiz $ FinalExam;
```

The length statement specifies the variable type (numeric or character) and length.

```
length variablename $ integervalue;
```

But there are times when the length statement is necessary when creating new character variables.

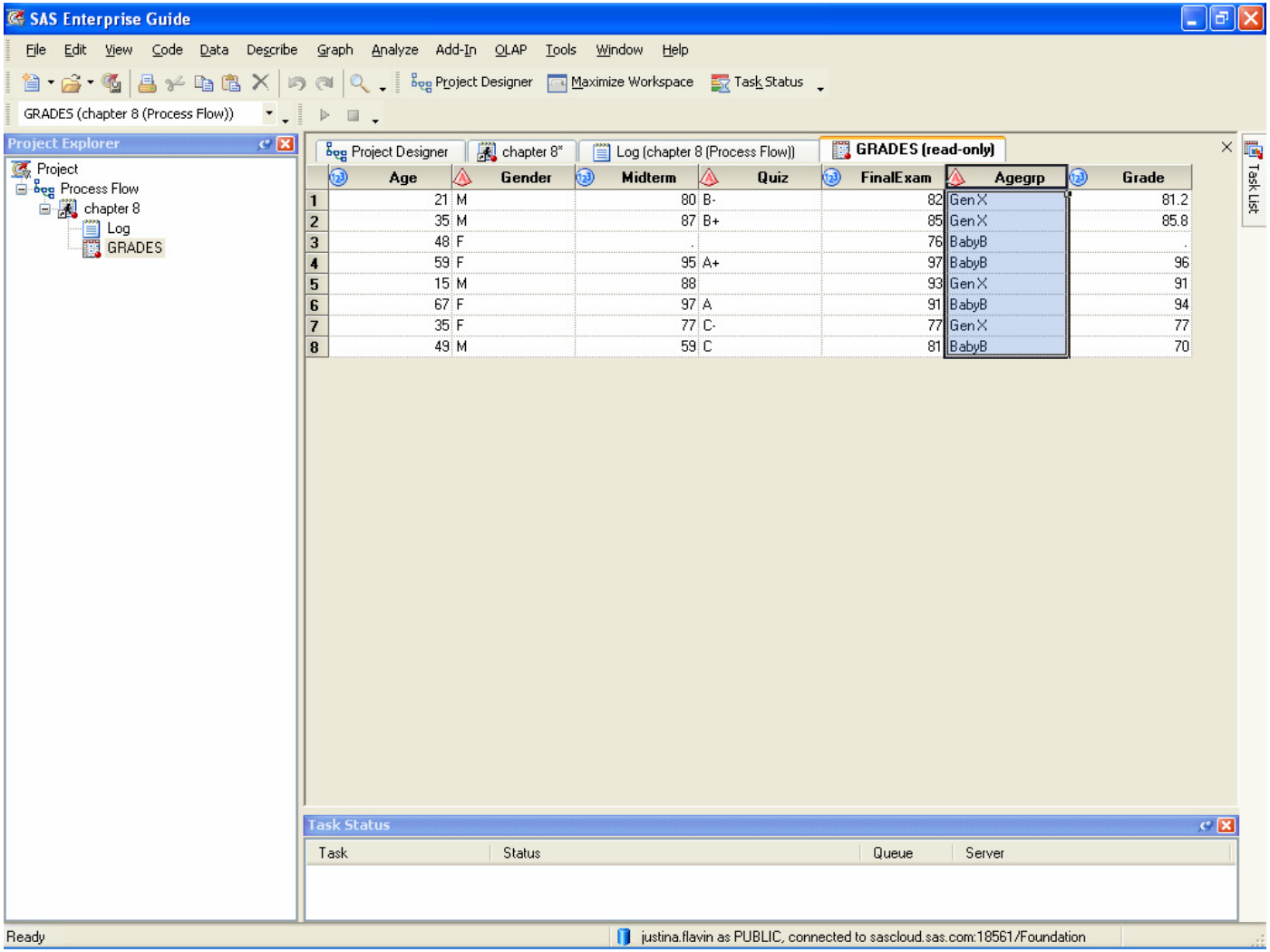
Here is the DO loop from my additional example in the Chapter 8 code.

```
if Age le 39 then do;
     Agegrp = 'Gen X';
     Grade = .4*Midterm + .6*FinalExam;
end;
else if Age gt 39 then do;
     Agegrp = 'BabyBoomers';
     Grade = (Midterm + FinalExam)/2;
end;
```

The only difference from the previous examples is that I have changed the text of the Agegrp variables.

Run the code and we will take a look at the resulting data set.

Here is the data set, GRADES




| | Age | Gender | Midterm | Quiz | FinalExam | Agegrp | Grade |
|---|---|---|---|---|---|---|---|
| 1 | 21 | M | 80 | B- | 82 | GenX | 81.2 |
| 2 | 35 | M | 87 | B+ | 85 | GenX | 85.8 |
| 3 | 48 | F | . | | 76 | BabyB | . |
| 4 | 59 | F | 95 | A+ | 97 | BabyB | 96 |
| 5 | 15 | M | 88 | | 93 | GenX | 91 |
| 6 | 67 | F | 97 | A | 91 | BabyB | 94 |
| 7 | 35 | F | 77 | C- | 77 | GenX | 77 |
| 8 | 49 | M | 59 | C | 81 | BabyB | 70 |

Notice the values of Agegrp.

The value “Gen X” is correct but “BabyBoomers” is truncated to the first 5 characters “BabyB”.

Why?

Because when SAS first encounters the new variable Agegrp in the code, it sets the attributes of the new variable based upon the value being assigned to it.

In this case, the first line of data has Age=21.

Thus, the first do loop is executed.

When SAS encounters the AgeGroup assignment statement, it creates a character variable of length 5 which is the proper length for the character string “Gen X”.

However, when processing occurs for the 4$^{th}$ data line, Age=48 so SAS executes the second do loop.

“BabyBoomers” is too long to properly fit into the Agegroup variable, so SAS truncates the value and assigns the first 5 characters “BabyB” to AgeGroup.

To fix this problem, the length statement can be added to the data step code to explicitly specify the proper variable length.

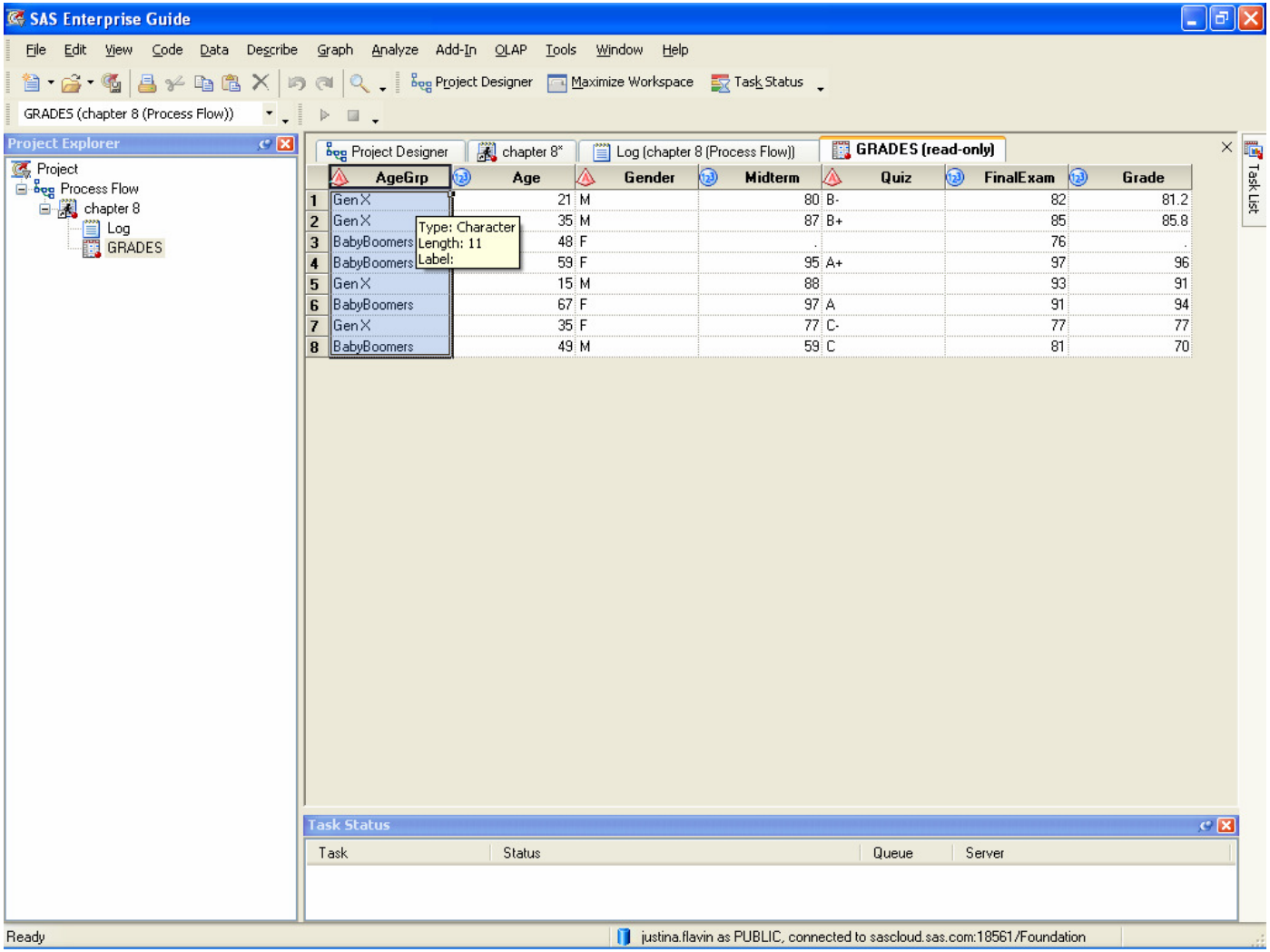
This means you need to determine the length of your longest data value and set the variable length to that value.

The length statement is usually placed at or near the top of the data step code, so that it is executed before values are added to the variables.

```
data grades;
  length AgeGrp $ 11;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/grades.txt" missover;
  input Age Gender $ Midterm Quiz $ FinalExam;
  if missing(Age) then delete;
  if Age le 39 then do;
    Agegrp = 'Gen X';
    Grade = .4*Midterm + .6*FinalExam;
  end;
  else if Age gt 39 then do;
    Agegrp = 'BabyBoomers';
    Grade = (Midterm + FinalExam)/2;
  end;
run;
```

Now the variable AgeGrp is long enough to hold the full length of all the values. In general, a good rule of thumb is to add a length statement when you are creating new character variables whose values will be of different lengths.

Notice too that now AgeGrp is physically the first variable in the data set. This is because SAS processes the length statement at the top of the data step and thus sets up the variable AgeGrp first before it encounters the rest of the variables on the input statement.



## 8.3 The Sum Statement

A sum statement can be used to create a counter or to accumulate running totals for a variable.

Missing values in arithmetic statements cause the result to also be a missing value.

The RETAIN statement is used to initialize a variable to a particular value.

As SAS processes each observation in the data set, it returns to the top of the data step code and initializes all variable values to missing.

Adding a RETAIN statement instructs SAS to retain the value from the previous iteration for any of the variables listed on the RETAIN statement.

Further, adding a value after the variable name sets the variable equal to that value the first time the code is executed.

Syntax:

```
retain variablename1 value1 variablename2 value2...variablename'n' value'n';
```

We will now look at a series of examples that use the sum statement in an attempt to write a program that creates a cumulative total for the variable Revenue.

Looking at Program 8-3, there are two code lines of interest:

```
input Day : $3.
      Revenue : dollar6.;
```

The colon ":" tells SAS to use the informat that is specified after the variable name, but to stop inputting when a delimiter is encountered.

So for the variable Day, SAS will read up to three characters, but if it encounters a blank BEFORE it has read three characters, it will output the characters prior to encountering the blank.

I have modified some of the data lines to illustrate this concept.

We are trying to create a new variable, Total, which is a running total of Revenue.

```
Total = Total + Revenue; /* Note: this does not work */
```

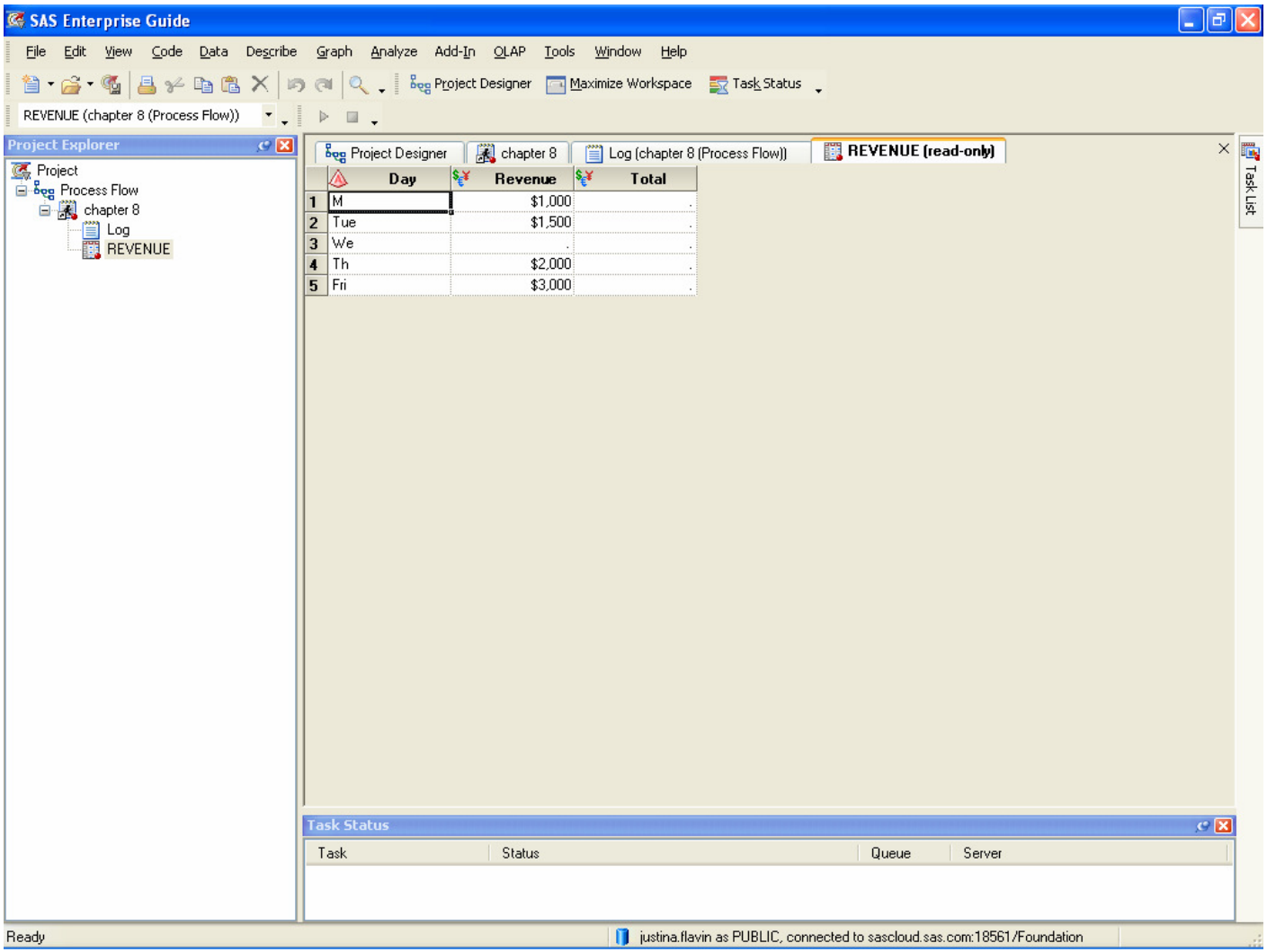
Run the code and we will look at the resulting data set.

As mentioned, this approach does not work.

SAS first encounters the variable Total in the assignment statement.

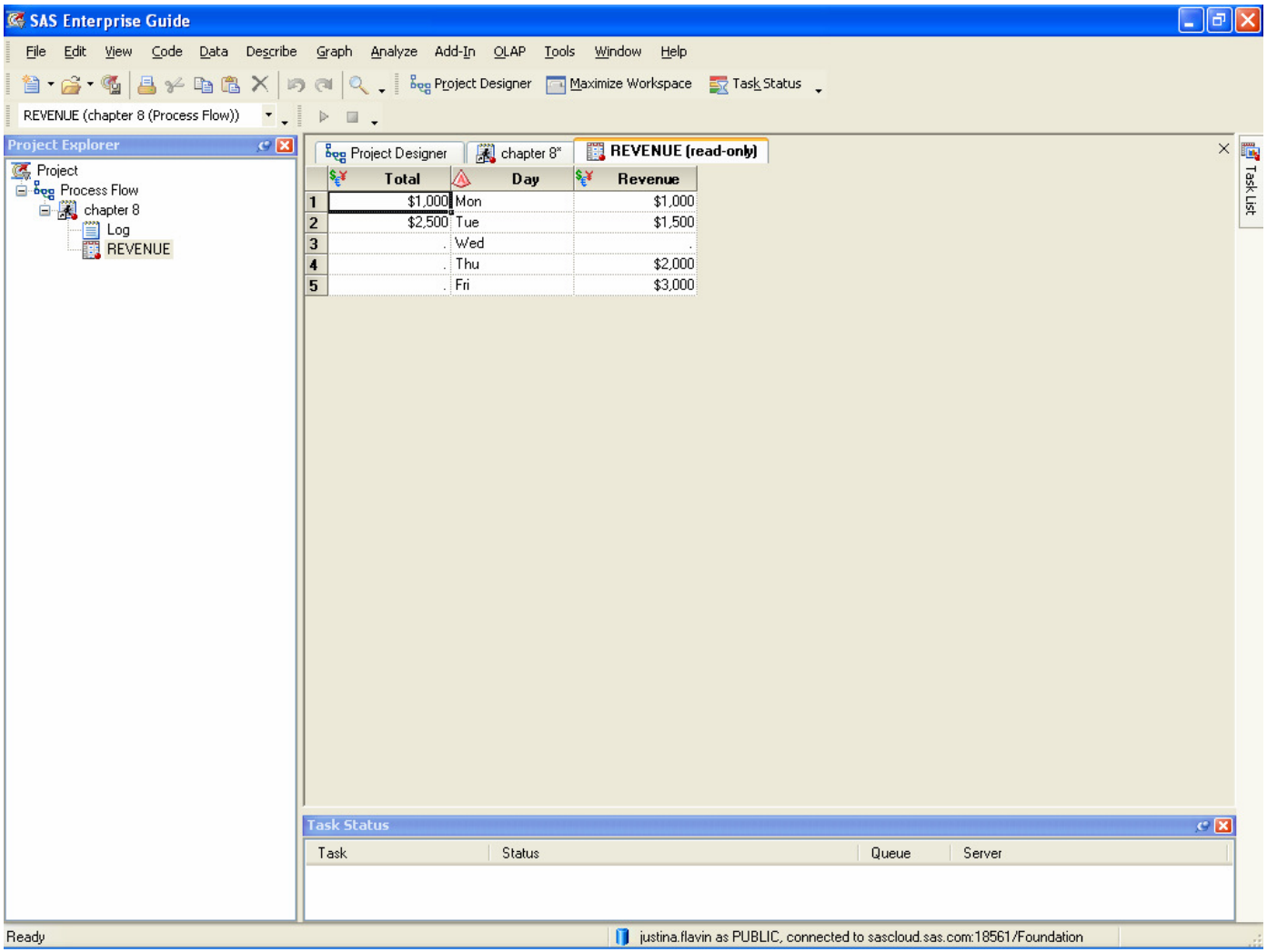
Since the variable does not exist, it is initially assigned a missing value.

We know that using a variable with a missing value in an assignment statement will yield a missing value for the result, so all of the values of Total are thus missing values.



Next in Program 8-4 a retain statement is added to the code.

```
retain Total 0;
```



Like the length statement, the retain statement is usually the first statement in the data step code.

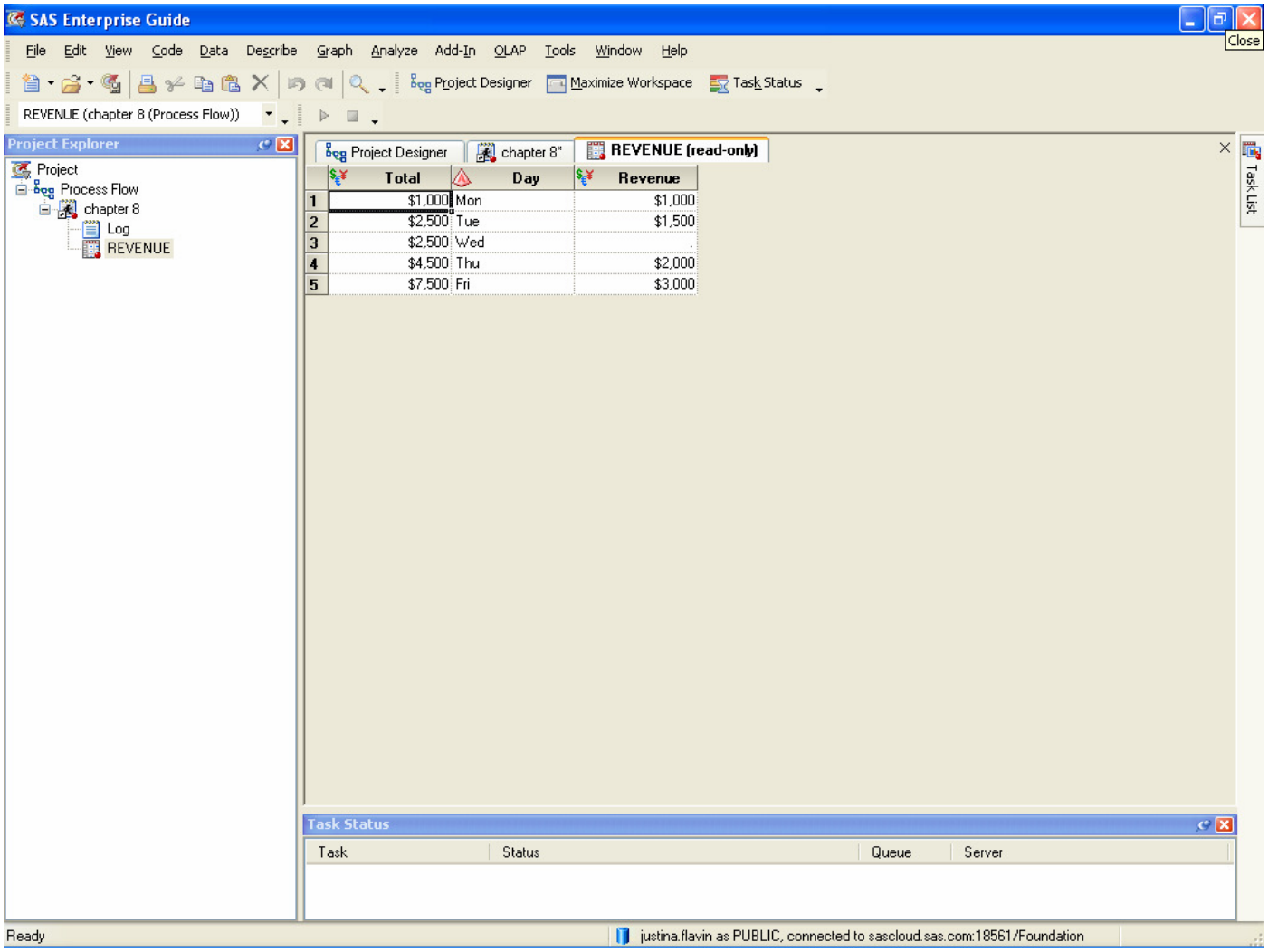
The variable Total is thus created first and in this example, it is initialized to the value of 0.

Again, this code does not give the correct result.

Because total is initially set to a value of 0, the results on the first two data lines are correct.

However, once the 3rd data line is encountered, we run into the same problem of missing values propagating missing results.

In Program 8-5 an IF-THEN statement is added so that the Total calculation is only performed if Revenue is not missing. Because of the retain statement, SAS retains the value of Total from each successive line. So the value of $2500 is retained from the 2nd line and carried down to the third line. Because of the IF-THEN statement, the Total calculation is not performed on the 3rd line. Now we have the desired result.



Program 8-6 illustrates an even easier way to obtain the desired result through the use of the sum statement:

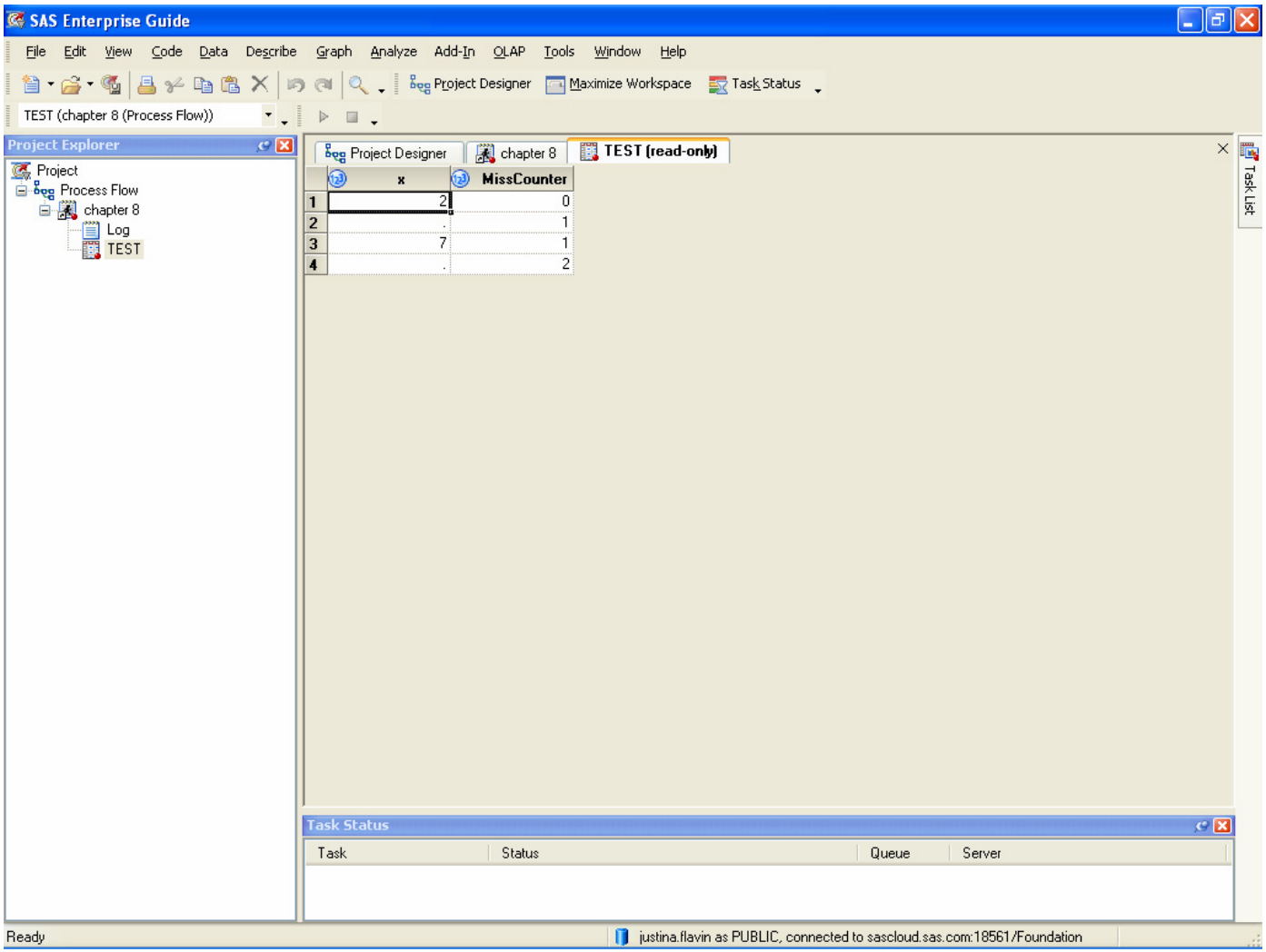
```
Total + Revenue;
```

When written this way, the retain statement is implied and not needed in the data stop code.

The variable Total is initialized to 0, retained, and missing values of Revenue are ignored so that the if-then condition is not necessary.

The sum statement is also useful for setting up a counter in the code. A counter can be used to count the number of occurances of some condition or value in the data. Program 8-7 illustrates the use of a counter to keep track of the number of missing values in the variable x.

Notice that the value of 1 is retained from observation 2 to 3 and then incremented on observation 4.



## 8.4 The Iterative DO Loop

The iterative DO loop is used to execute the same code multiple times, making use of a counter variable that is incremented each time the loop is executed.

These examples will also illustrate the use of the OUTPUT statement.

The OUTPUT statement is used to output an observation to the output data set during the execution of the data step.

In these examples we invest $100 for 3 years at 3.75% interest and wish to determine the amount of money we will have at the end of each year.

First we will look at a program that works without using an iterative do loop.

```
*Program 8-8 Program without iterative loops - page 125;
data compound;
   Interest = .0375;
   Total = 100;

   Year + 1;
   Total + Interest*Total;
   output;

   Year + 1;
   Total + Interest*Total;
   output;

   Year + 1;
   Total + Interest*Total;
   output;

   format Total dollar10.2;
run;
```

Notice that there are no data lines that are read into the program.

First the variables Interest and Total are created and assigned initial values.

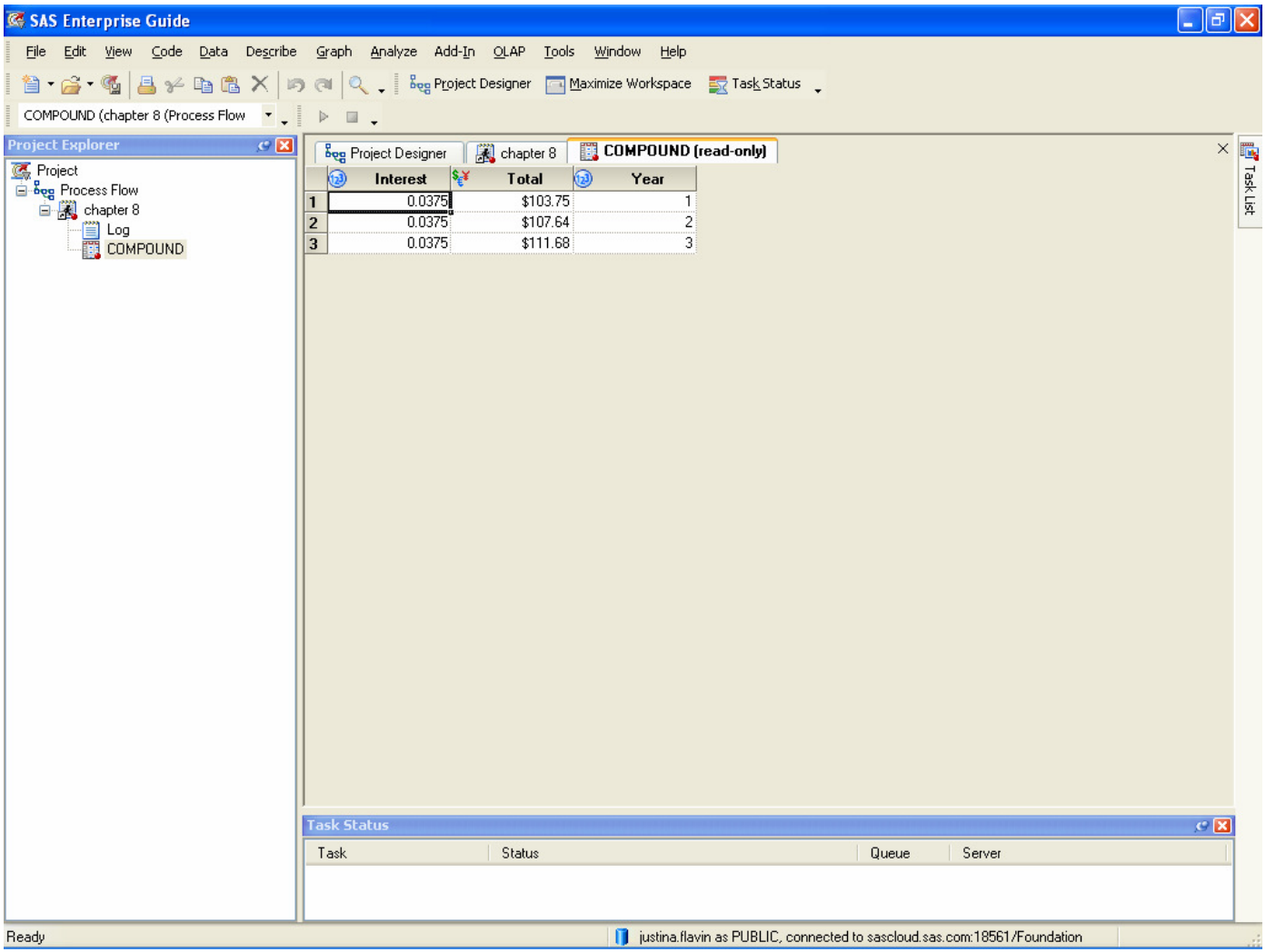
Next, the variable Year is created using a sum statement.

Finally, the Total is calculated using a sum statement.

The output statement causes SAS to write an observation with all the variables to the data set compound.

Then the same statements are executed again twice, resulting in two more observations being written to the data set.

Here is the data set COMPOUND



Since we are executing the same code lines over & over, we can simplify this code with the use of an iterative do loop.

```
*Program 8-9 Demonstrating in iterative DO loop - page 126;
data compound;
   Interest = .0375;
   Total = 100;
   do Year = 1 to 3;
      Total + Interest*Total;
      output;
   end;
   format Total dollar10.2;
run;
```

With the iterative do loop, the variable Year takes on the values 1,2,3.

Year is set to 1 and the statements within the do loop are executed.

When the end statement is executed, SAS returns to the top of the do loop, sets Year=2 and executes the statements inside the do loop again.

After executing the statements for Year=3, SAS exits the do loop and continues processing the rest of the statements in the data step.

Program 8-10 presents another interesting use of a do loop to create a table of squares and square roots.

```
*Program 8-10 Using an iterative Do loop to make a table of squares and square roots - page 127;
data table;
   do n = 1 to 10;
      Square = n*n;
      SquareRoot = sqrt(n);
      output;
   end;
run;
```

Again in this example, there is no data to be read into the data step.
The data are created based on the calculations within the do loop.

This example also makes use of the square root function, sqrt.
In this example, the sqrt function calculates the square root of n and then assigns that value to the variable SquareRoot.

We will look at functions in much more detail in Chapters 11 and 12.

Finally, Program 8-11 illustrates the use of a do loop to graph a quadratic equation.

This example illustrates the use of the BY option on the do statement which allows for increments other than 1.

In this example, X ranges from -10 to 10 using increments of 0.01, resulting in a large data set of 2001 observations.

X**n is X raised to the nth power, so in the calculation of Y, X**3 is X cubed and X**2 is X squared.

Without going into much detail here, PROC GPLOT is a procedure to generate plots of 2 or 3 variables.

It is part of the SAS/GRAPH module. The symbol statement is used in this example to control the attributes of the line that is drawn.

```
*Program 8-11 Using an iterative DO loop to graph an equation - page 128;
data equation;
   do X = -10 to 10 by .01;
      Y = 2*x**3 - 5*X**2 + 15*X -8;
      output;
   end;
run;

symbol value=none interpol=sm width=2;
title "Plot of Y versus X";
proc gplot data=equation;
   plot Y * X;
run;
```

## 8.5 Other Forms of an Iterative DO Loop

Character values can also be used with a do loop.

Multiple do loops can be embedded.

The “@” character at the end of an INPUT statement is called a trailing @ sign.

It is used to instruct SAS to “hold on” to the current input data line and continue to read data from that line on the next iteration of the data step, rather than moving to the next line of data.

Syntax:

```
input variablename1 variablename2...variablename'n' @;
```

Program 8-12 illustrates the use of embedded do loops and the use of character values in a do loop.

```
*Program 8-12 Using character values for DO loop index values - page 130;
data easyway;
   do Group = 'Placebo','Active';
      do Subj = 1 to 5;
         input Score @;
         output;
      end;
   end;
datalines;
250 222 230 210 199
166 183 123 129 234
;
run;
```

In this example, the first line of data corresponds to Placebo and the second line of data is Active.

SAS begins by assigning Group=Placebo. Next the second do loop is entered and executed 5 times. Each time the input line is read, SAS reads in the value of Score, then “remembers” its position on the data line. At the next iteration of the input statement, the next value of Score is read and again the position is retained. This continues until no more data values are found on the data line.

After the do loop is executed for Subj=5, SAS exits the inner do loop. Then it returns to the outer do loop, sets Group=Active and re-executes the inner do loop so that the second line of data is read.

The number of observations in the EASYWAY data set is 10, since there are 2 values of Group and 5 values of Score, so 2*5=10.

## 8.6 DO WHILE and DO UNTIL Statements

DO WHILE and DO UNTIL loops are used when data values are used as the criteria for code execution.

A DO-UNTIL loop continues executing until a condition is met.

In Program 8-13, the do loop is executed until the value of Total reaches or exceeds 200.

The do loop is executed 19 times.

At the end of the 19th iteration, Total=201.27, so the condition (Total ge 200) is true and the loop is exited.

```
*Program 8-13 Demonstrating a DO UNTIL loop - page 131;
data double;
   Interest = .0375;
   Total = 100;
   do until (Total ge 200);
      Year + 1;
      Total = Total + Interest*Total;
      output;
   end;
   format Total dollar10.2;
run;
```

A DO-UNTIL loop always executes at least once.

This is because the UNTIL condition is tested at the END of the loop. This is illustrated in Program 8-14.

```
 Total = 300;
do until (Total gt 200);
```

Even though Total is initially set to 300, the do until loop executes once.

When SAS reaches the end statement, it checks the condition, determines that Total is greater than 200 and exits the loop.

A DO-WHILE loop continues executing while a condition is true.

Program 8-15 illustrates the use of a DO WHILE loop which accomplishes the same task as the previous DO-UNTIL code.

```
do while (Total le 200);
      Year + 1;
      Total = Total + Interest*Total;
      output;
end;
```

Unlike the UNTIL condition, the WHILE condition is tested at the TOP of the loop.

Thus a DO-WHILE loop may not ever execute. This is illustrated in Program 8-16.

```
Total = 300;
do while (Total lt 200);
```

Because the initial value of Total is 300, `do while (Total lt 200)` is never true, so the do while loop is never executed and the resulting data set has 0 observations.

## 8.7 A Caution When Using DO UNTIL Statements

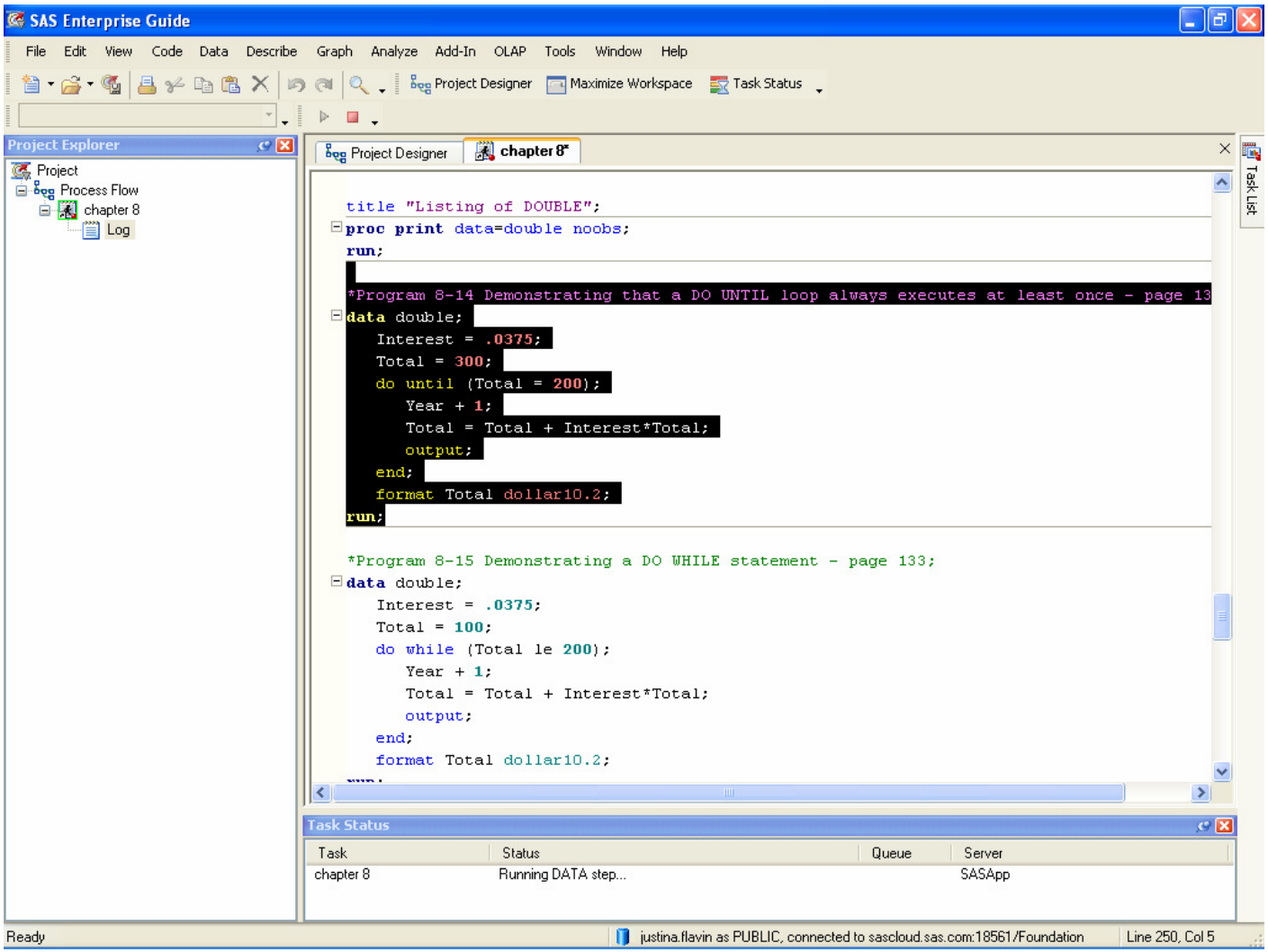
When using DO-UNTIL statements, it is possible to write your code in such a way that you create an infinite loop.

If the condition is never met, once SAS enters the loop, it continues to execute over and over infinitely so code execution never ends.

If this happens, you must terminate the process as illustrated on the next slides.

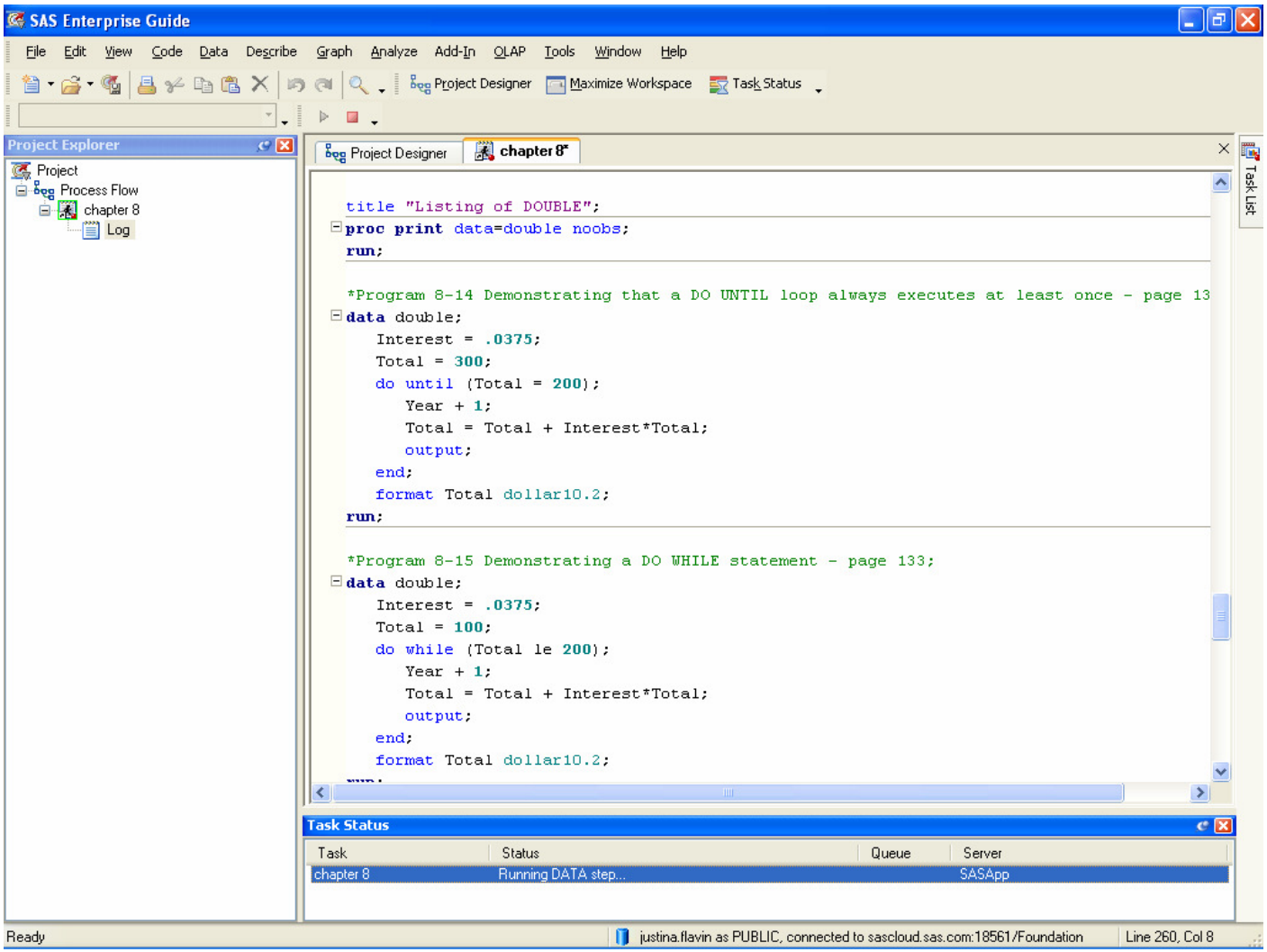
A less elegant way to terminate the process is to close your SAS session and open a new session.

In Program 8-14, the do-until loop is changed to `do until (Total = 200);` and the code is executed. The Status line at the bottom becomes Running DATA step…



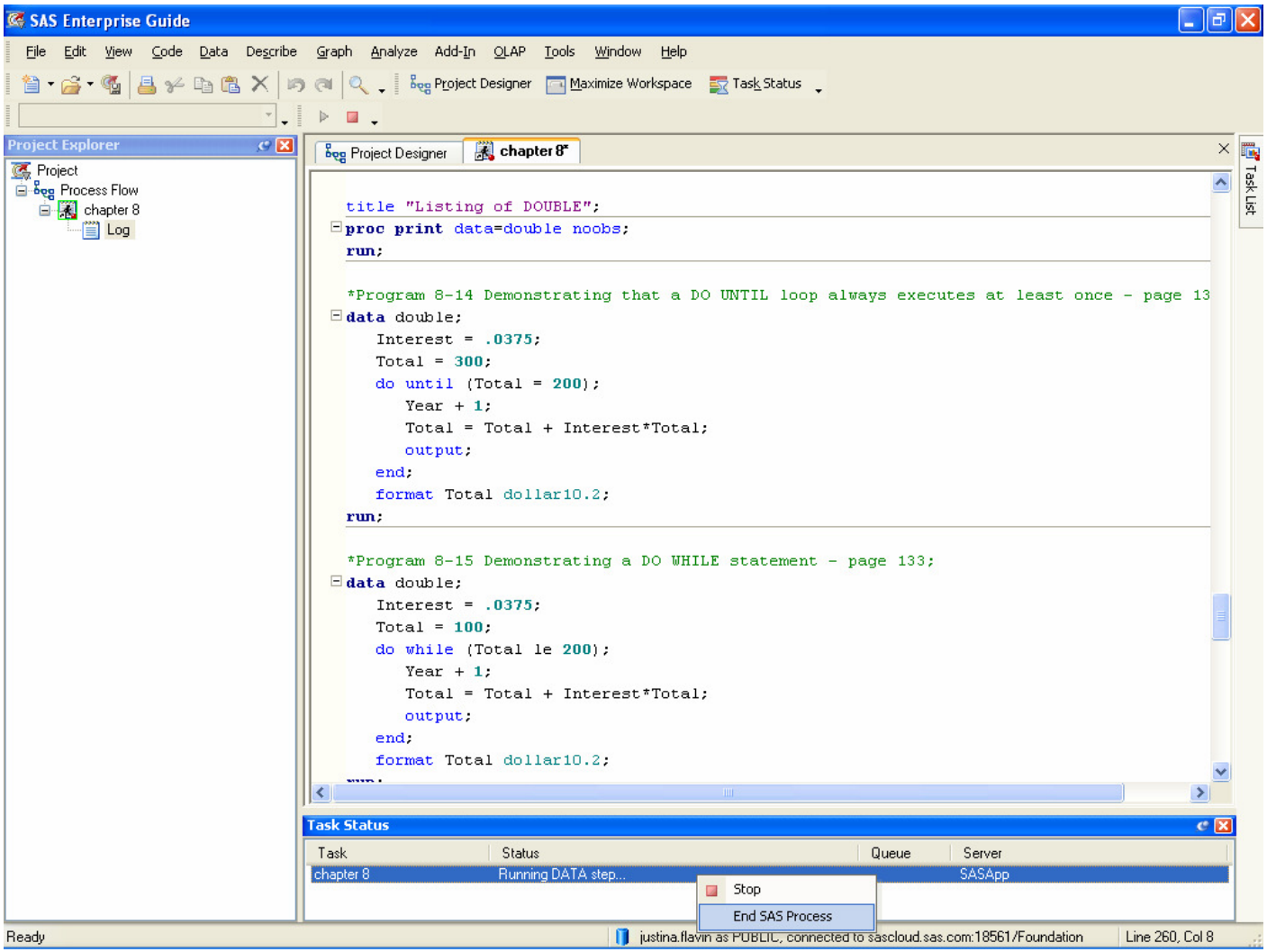

 

To stop execution, highlight the status row at the bottom of the screen.

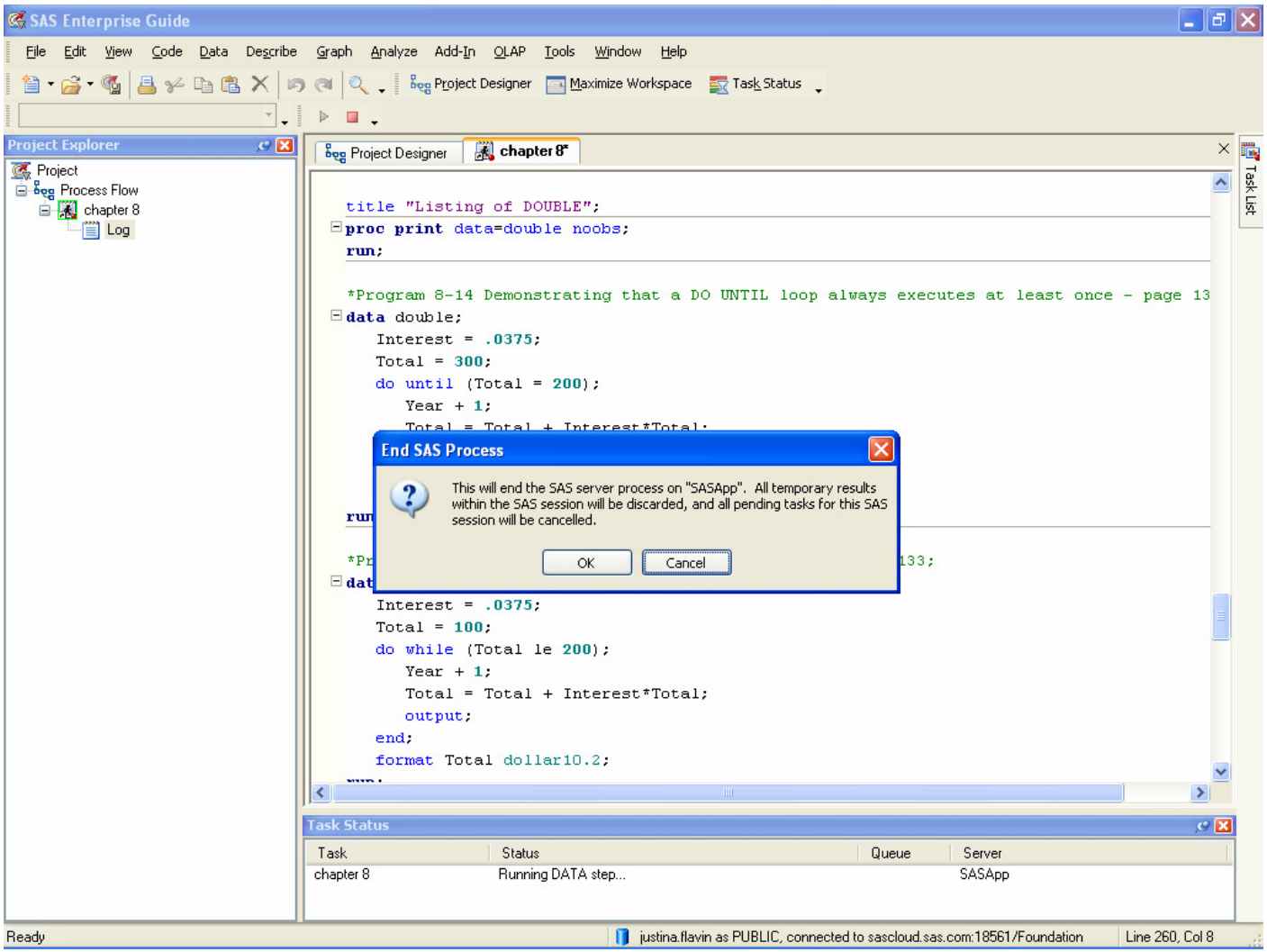


Right click and select ‘End SAS Process’

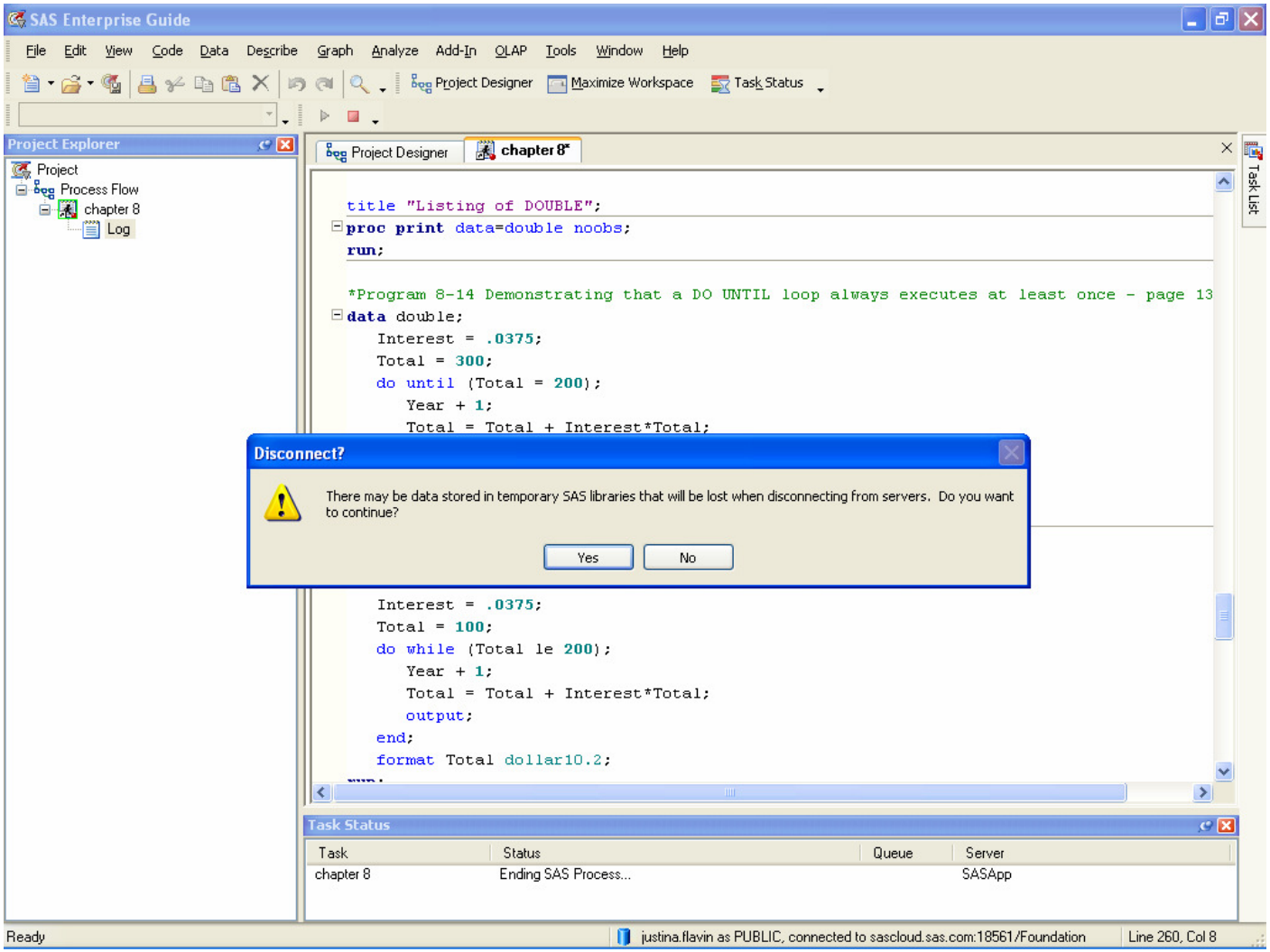


Select OK



Select Yes



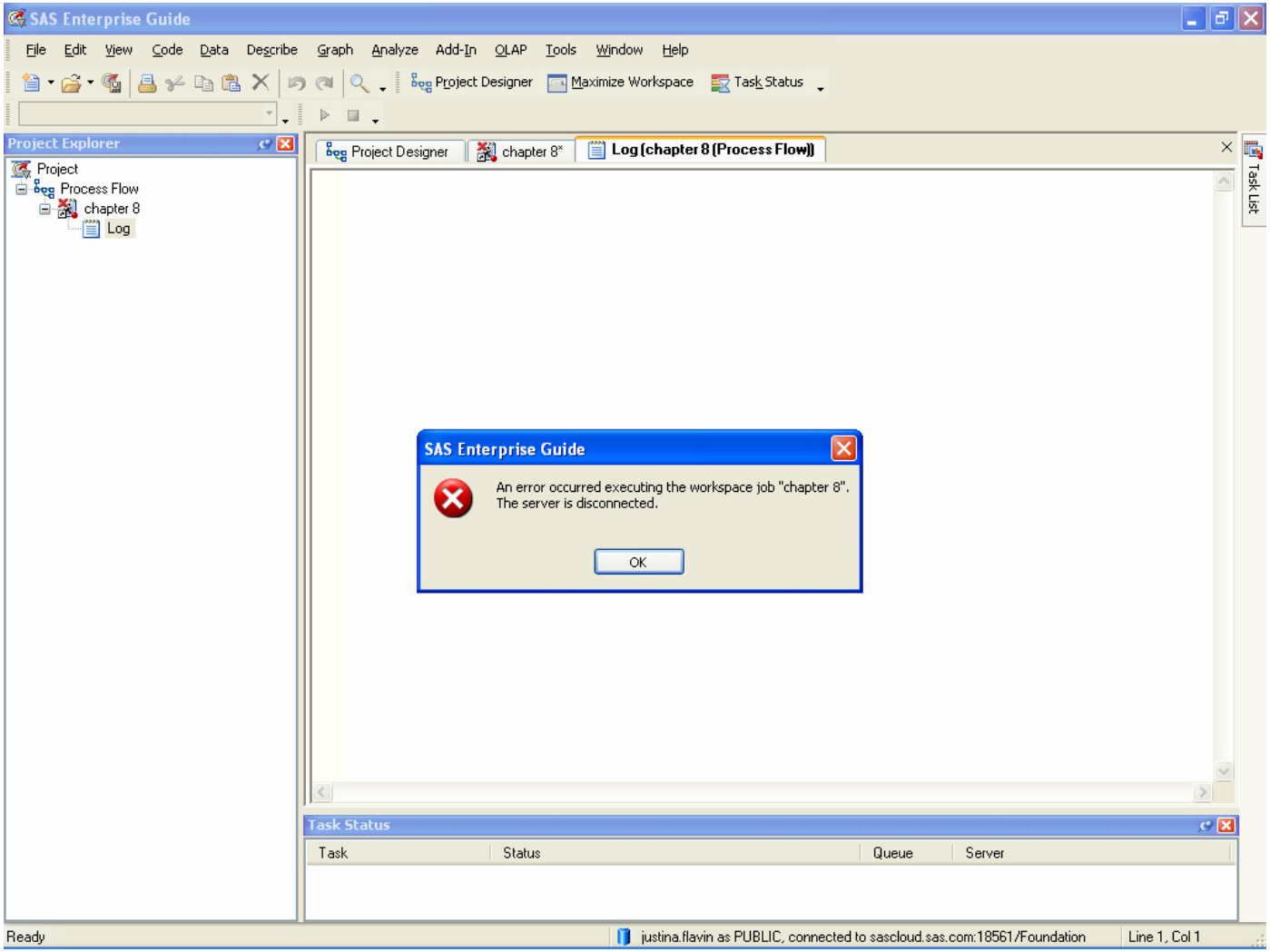

 

Select OK.

Now the infinite loop has been terminated and you can resume working.

Notice the red X on Chapter 8 under Process Flow.

This indicates that SAS encountered an error or problem while running the code.



Program 8-17 (modified) illustrates a way to prevent creating an infinite loop when using an UNTIL statement, by adding an additional condition to the do loop:

```
do Year = 1 to 100 until (Total = 200);
      Total = Total + Interest*Total;
      output;
end;
```

Now if the condition of Total = 200 is never met, SAS simply executes the loop 100 times and then exits.

Run this code and the resulting data set will have 100 observations.

## 8.8 LEAVE and CONTINUE Statements

The LEAVE statement within a DO loop is used to exit the loop and terminate further execution of the code.

The CONTINUE statement causes the termination of execution of the statements following the CONTINUE statement within the loop, but, rather than exiting the loop, execution is returned to the top of the loop.

Program 8-18 (modified) illustrates the use of the LEAVE statement.

When the value of TOTAL becomes greater than or equal to 200, OR when the value of Year=5 then the DO loop is exited.

In this modified example, Year=5 before Total becomes ge 200, so execution stops after 5 iterations of the do loop (and hence 5 observations are written to the data set.)

```
*Program 8-18 Demonstrating the LEAVE statement - page 135;
data leave_it;
   Interest = .0375;
   Total = 100;
   do Year = 1 to 5;
      Total = Total + Interest*Total;
      output;
      if Total ge 200 then leave;
   end;
   format Total dollar10.2;
run;
```

Program 8-19 illustrates the use of the CONTINUE statement.

When `if Total le 150 then continue;` is evaluated, if the result is true (that is, Total is less than 150), the statements following that statement in the loop are passed over and not executed.

This causes the output data set to only consist of observations having Total ge 150.

When SAS gets to the end statement, control returns to the top of the do loop.

So the continue statement only affects “what happens” inside the do loop.

It has no affect on the actual number of times the do-until loop executes.

```
*Program 8-19 Demonstrating a CONTINUE statement - page 136;
data continue_on;
   Interest = .0375;
   Total = 100;
   do Year = 1 to 100 until (Total ge 200);
      Total = Total + Interest*Total;
      if Total le 150 then continue;
      output;
   end;
   format Total dollar10.2;
run;
```

The last example is a modification of Program 8-19 that illustrates the use of a retain statement and a counter to accomplish the same task.

The assignment statements for interest and total are replaced by a retain statement with the initial variable values.

The assignment statement for updating Total in the do loop is replaced with a sum statement.

The resulting data set is identical.

In terms of program and code efficiency, use of a retain statement is more efficient than use of an assignment statement because in most data steps, the retain statement is only executed once, whereas an assignment statement is executed multiple times.

```
/* Program 8-19 - using a retain statement and counter */
data continue_on2;
   retain interest .0375 total 100;
   do Year = 1 to 100 until (Total ge 200);
      Total + Interest*Total;
      if Total le 150 then continue;
      output;
   end;
   format Total dollar10.2;
run;
```