SAS Programming (BIOL-4V190)

Chapter 11

Working with Numeric Functions

11.1 Introduction

Functions are built-in routines that are provided as part of the SAS Programming language.

A few functions have already been presented in previous lectures.

Using functions can greatly simplify your code since in many cases it prevents you from having to write tedious or complex code.

Functions fall into two categories, character and numeric.

Character functions are used for manipulating character variables while numeric functions are used with numeric variables.

A subcategory of numeric functions are date and time functions which are used with date and time variables.

This chapter will cover some of the most commonly used numeric functions.

There are many more functions available than will be covered in this course.

The best way to learn about the functions that are available is to browse through the functions section on the online documentation.

Information about functions in the online documentation can be found at:
http://support.sas.com/documentation/onlinedoc/91pdf/index.html

SAS OnlineDoc 9.1 for the Web -> Base SAS -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Functions and CALL Routines

The Functions and CALL Routines by Category link is especially helpful.

11.2 Functions That Round and Truncate Numeric Values

The **int** function returns the integer value

Syntax:
INT(*argument*)

argument can be a variable name, constant, or any SAS expression, including another function

The **round** function rounds the first argument to the nearest multiple of the second argument, or to the nearest integer when the second argument is omitted.

Syntax:

ROUND(*argument* <,*rounding-unit*>)

rounding-unit is a positive, numeric constant, variable, or expression that specifies the rounding unit

```
*Program 11-1 Demonstrating the ROUND and INT truncation functions - page 191;
data truncate;
  input Age Weight;
  Age = int(Age);
  WtKg = round(2.2*Weight,.1);
  Weight = round(Weight);
datalines;
18.8 100.7
25.12 122.4
64.99 188
;
run;
```

In this example, Age is read in on the input line, but on the following line, the int function is applied to the variable, so only the integer portion of age will be stored in the data set.

SAS computes WtKg by calculating 2.2*Weight first and then rounding that result to the closest tenth or 0.1

Like Age, Weight is read in on the input line, then it is rounded to the nearest integer since no rounding unit was supplied.

The resulting data set is shown on page 191.

11.3 Functions That Work with Missing Values

missing() is a function that can be used with both numeric and character data.
It determines if a variable value is missing or not missing.
First we will look at an example that does not use the missing function.

```
*Program 11-2 Testing for numeric and character missing values (without the missing function)
 - page 192;
data test_miss;
   set learn.blood;
   if Gender = ' ' then MissGender + 1;
   if WBC = . then MissWBC + 1;
   if RBC = . then MissRBC + 1;
   if Chol lt 200 and Chol ne . then Level = 'Low ';
   else if Chol ge 200 then Level = 'High';
run;
```

In this example we wish to create counter variables to add up and store the number of missing values in several variables.

To specify a missing value for a character variable, use two quotes with or without a blank between them
(i.e. ‘’ or ‘ ‘)

To specify a missing value for a numeric variable, use a period.

As SAS processes each record in the input data set, if it encounters a missing value for Gender, WBC, or RBC the respective counter variable is incremented.

The values for MissGender MissWBC MissRBC in the last observation of test_miss contain the total number of missing values for each of those variables.

This next example illustates how to rewrite the code using the missing function to obtain the same result.

```
*Program 11-3 Demonstrating the MISSING function - page 192;
data test_miss;
   set learn.blood;
   if missing(Gender) then MissGender + 1;
   if missing(WBC) then MissWBC + 1;
   if missing(RBC) then MissRBC + 1;
   if Chol lt 200 and not missing(Chol) then Level = 'Low ';
   else if Chol ge 200 then Level = 'High';
run;
```

This example also illustrates the use of the not operator: not missing(Chol)

This can be interpreted as “value of Chol is not missing”

*****Coding Missing Values in Numeric Data*****

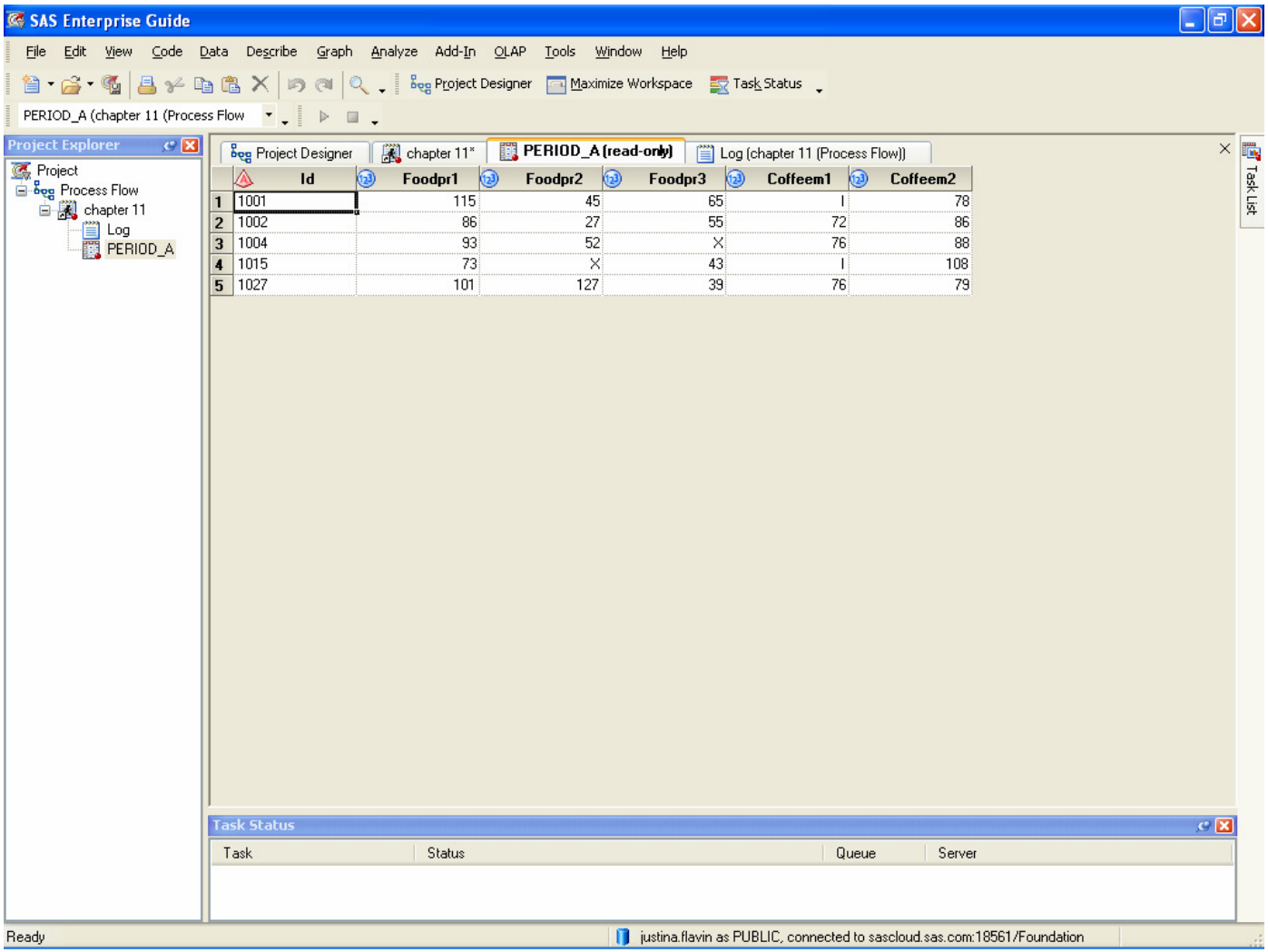
Missing values in numeric data can be flagged two different ways.
This enables you to customize the display of the missing values in output.
There are two methods for doing this.

1st method: Use a **MISSING** statement in the data step and specify the values which will appear in the data when a missing value is encountered. The method allows you to specify up to 26 different missing value codes (one for each letter of the alphabet).

```
/* 1st method: Use a MISSING statement */
data period_a;
  missing X I;
  input Id $4. Foodpr1 Foodpr2 Foodpr3 Coffeem1 Coffeem2;
  datalines;
1001 115 45 65 I 78
1002 86 27 55 72 86
1004 93 52 X 76 88
1015 73 X 43 I 108
1027 101 127 39 76 79
  ;
run;
```

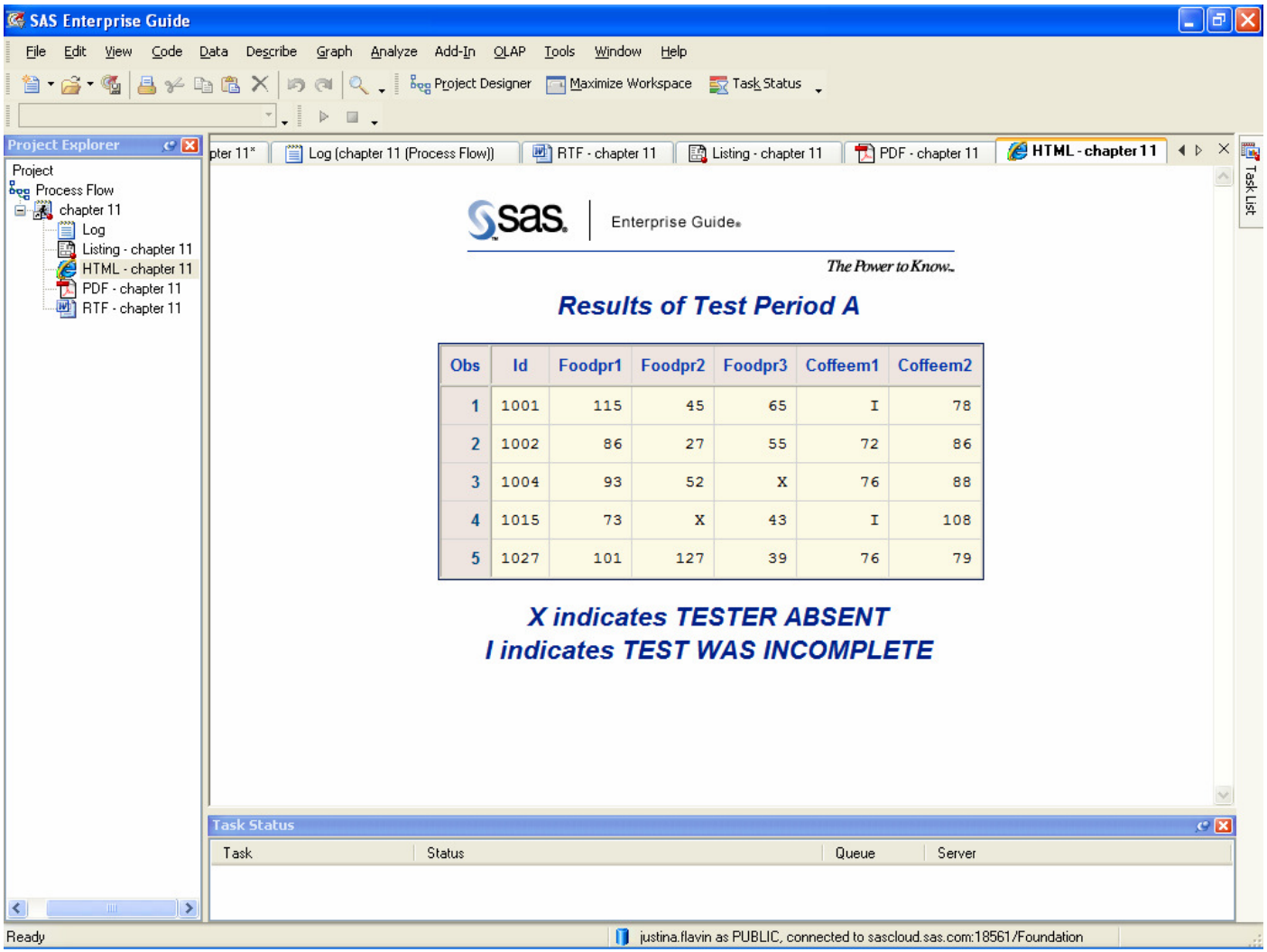
In this example, the missing statement is used to inform SAS that there will be two types of missing numeric values which will be coded in the data as X and I.
Note that there are missing values on the 1st, 3rd, and 4th data lines.

Notice that the missing values appear as the character values assigned in the data step.



	Id	Foodpr1	Foodpr2	Foodpr3	Coffeem1	Coffeem2
1	1001	115	45	65	I	78
2	1002	86	27	55	72	86
3	1004	93	52	X	76	88
4	1015	73	X	43	I	108
5	1027	101	127	39	76	79

A footnote can be used to provide an explanation for the missing value categories.



Another way to display the missing value categories is through the use of formats.

To create a format for numeric missing values, place a period before the missing value character.

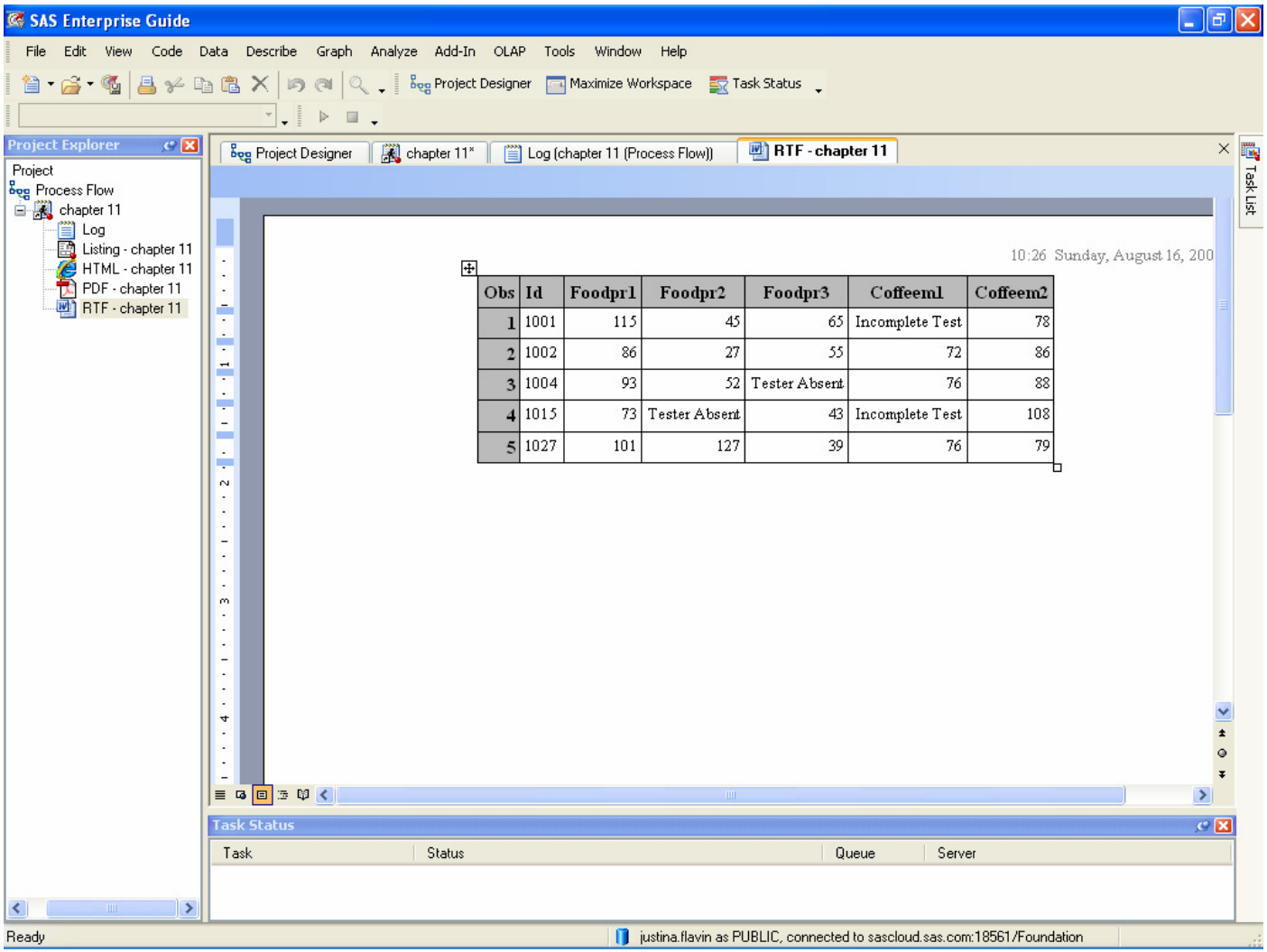
numeric is an automatic variable that refers to ALL numeric variables in a data set.

There is also an automatic variable called _character_ which refers to ALL character variables in a data set.

`format _numeric_ missing.;` instructs SAS to apply the format named 'missing' to all numeric variables in the data set.

```
* display missing codes by creating a format;
proc format;
  value missing .x='Tester Absent'
                .i='Incomplete Test'
                .c='Result not Recorded'
                  ;
run;

proc print;
  format _numeric_ missing.;
run;
```

Now when the data are printed, the formatted values are displayed rather than the character codes.



Obs	Id	Foodpr1	Foodpr2	Foodpr3	Coffeem1	Coffeem2
1	1001	115	45	65	Incomplete Test	78
2	1002	86	27	55	72	86
3	1004	93	52	Tester Absent	76	88
4	1015	73	Tester Absent	43	Incomplete Test	108
5	1027	101	127	39	76	79

2nd method: Use an **OPTIONS MISSING='*value*'** statement before coding the data step. '*value*' is a single character enclosed in quotes. This method will flag ALL missing values in ALL numeric variables in your data set.

This method is easier if you only have one type of missing value. But if there are different types of missing values and you wished to distinguish between them, then the previous method must be used.

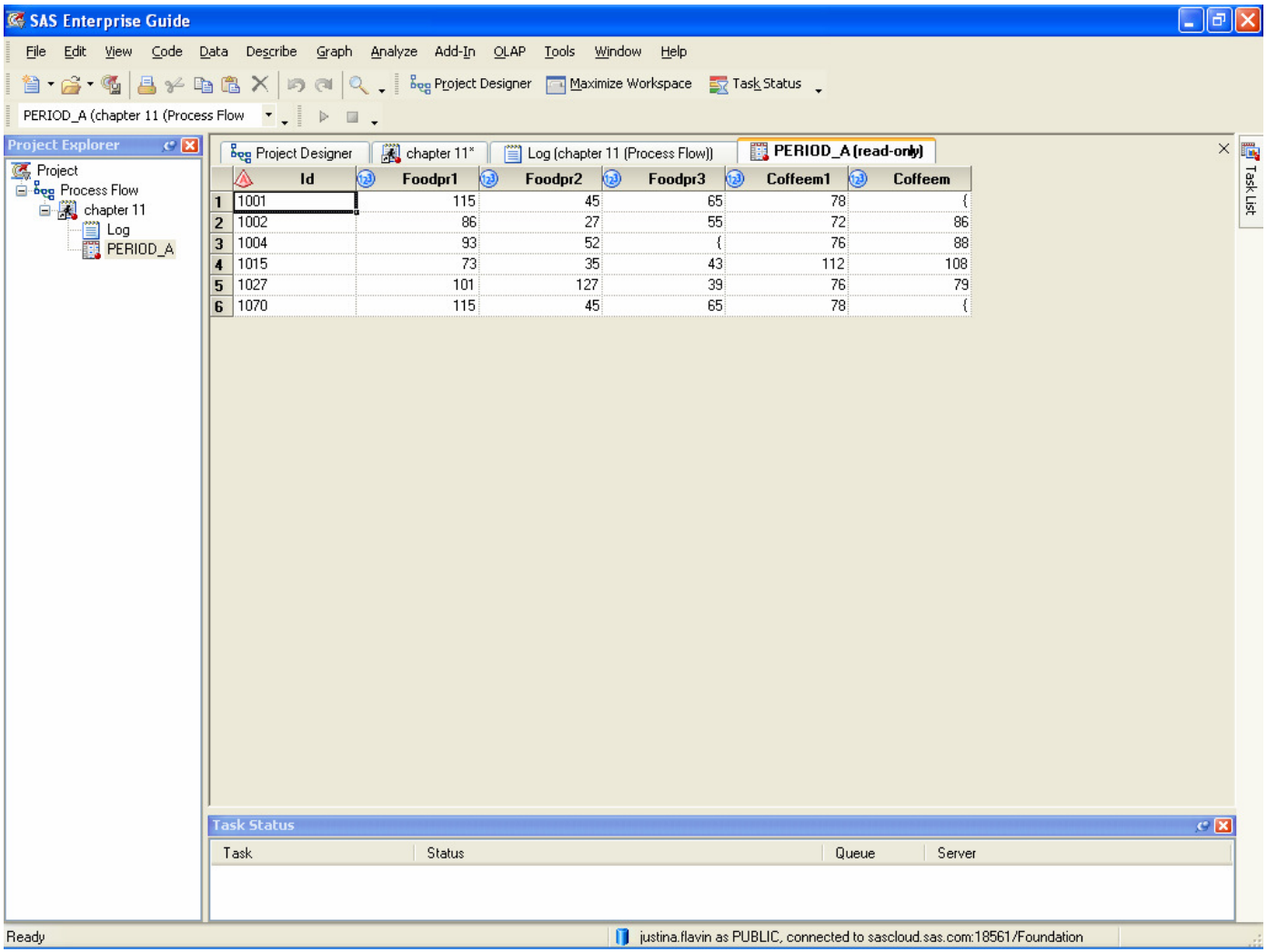
```
/* 2nd method: Use an OPTIONS MISSING='value' statement */
options missing="{";

data period_a;
  input Id $4. @6 Foodpr1 3. @10 Foodpr2 3. @14 Foodpr3 2. @17 Coffeem1 3. @21 Coffeem 3.;
datalines;
1001 115  45 65  78
1002  86  27 55  72  86
1004  93  52     76  88
1015  73  35 43 112 108
1027 101 127 39  76  79
1070 115  45 65  78   .
  ;
run;
```

Since the options statement is a global statement, it is typically placed outside the data step code. In this example, the character '{' is specified as the character which will be used to display the missing values.

Notice that in the lines of data there is a missing value at the end of the first and last data lines. The first data line is just blank while the last data line has a period to denote the missing value. SAS interprets both of these as missing values and will apply the missing value character to both.

A caveat when using this method is that since the options statement is a global statement, after the statement is executed, SAS will apply this missing value character to every missing numeric value that it encounters in all subsequent code.

The “{“ character replaces the missing values.



11.4 Setting Character and Numeric Values to Missing

This section is omitted.

11.5 Descriptive Statistics Functions

There is a group of functions that can provide descriptive statistics on the variables in an observation (or row of data). A complete list of these is available in the online documentation in the **Descriptive Statistics** section of the **Functions and CALL Routines by Category**.

The functions that will be illustated in the examples are:
n – returns the number of nonmissing values
mean – returns the mean
std – returns the standard deviation
max – returns the maximum value
min – returns the minimum value
largest – returns the nth largest value

With some functions, the argument of a function can be created by using the keyword OF in front of a series of variables written using variable list notation. Note: This only works with some functions.

Syntax:

```
functionname (OF variablename1-variablename'n')
```



```
*Program 11-4 Demonstrating the N, MEAN, MIN, and MAX functions - page 194;
data psych;
  input ID $ Q1-Q10;
  if n(of Q1-Q10) ge 7 then Score = mean(of Q1-Q10);
  MaxScore = max(of Q1-Q10);
  MinScore = min(of Q1-Q10);
  meanscore = mean(Q1,q2,q5);
  std = std(q2-q5);
datalines;
001 4 1 3 9 1 2 3 5 . 3
002 3 5 4 2 . . . 2 4 .
003 9 8 7 6 5 4 3 2 1 5
;
run;
```

In this example, n(of Q1-Q10) is the number of nonmissing values in the variables Q1 through Q10.

If there are 3 or less missing values, then Score is calculated as the mean of the variables Q1 through Q10.

The argument of these functions can also be specified by listing the variables and separating them by a comma inside the parentheses as shown for the variable meanscore.

```
*Program 11-5 Finding the sum of the three largest values in a list of variables - page 195;
data three_large;
  set psych(keep=ID Q1-Q10);
  SumThree = sum(largest(1,of Q1-Q10),
                 largest(2,of Q1-Q10),
                 largest(3,of Q1-Q10));
run;
```

This example illustrates the concept of embedded functions (or a function within a function).
With embedded functions, the innermost function is evaluated first.
This result is then passed on to the outer function.

In this example, the sum function is the outer function and the largest functions are the inner functions.

For the first row of data, largest(**1**,of Q1-Q10) is the 1st largest value of the variables Q1-Q10.
This would be the value of 9 in Q4.

largest(2,of Q1-Q10) is the 2nd largest value of the variables Q1-Q10.
This would be the value of 5 in Q8.

largest(3,of Q1-Q10) is the 3rd largest value of the variables Q1-Q10.
This would be the value of 4 in Q1.

So SumThree = sum(largest(**1**,of Q1-Q10),
largest(**2**,of Q1-Q10),
largest(**3**,of Q1-Q10));
evaluates to sum(9,5,4) or 18

11.6 Computing Sums within an Observation

SAS treats missing values differently in assignment statements and functions.

When a missing value is encountered in an assignment statement expression, the result will be a missing value.

When a missing value is encountered in a function, any missing values are ignored and the result is calculated by using only the nonmissing values

This is important to remember when you are writing your code.

You must decide which way you wish to handle the missing values and then write you code appropriately.

```
*Program 11-6 Using the SUM function to compute totals - page 197;
data sum;
   set learn.EndOfYear;
   Total = sum(0, of Pay1-Pay12, of Extra1-Extra12);
   tot_by_fcn=sum(of pay1-pay4);
   tot_by_assignment=pay1+pay2+pay3+pay4;
run;
```

In this example, for observation 1, Total=sum(0,34800,1000)=35800

For observation 2, Total=sum(0, ., .)=0

For observation 3, Total=sum(0,28300 , .)=28300

tot_by_fcn and tot_by_assignment illustrate the difference in the results when using a function and using an assignment statement with data having missing values.

Observation 1 has a missing value for Pay4.

Using a function, the missing value is ignored and tot_by_fcn = 9400.

Using an assignment statement, the missing value causes the result to also be missing for tot_by_assignment

There are no missing values for Pay1-Pay4 in observation 3, so the results are the same for both tot_by_fcn and tot_by_assignment (9200).

11.7 Mathematical Functions

There are a large number of mathematical functions available.
A complete list may be found in the online documentation in the **Mathematical** section of the **Functions and CALL Routines by Category.**

The mathematical functions used in the examples are:
abs – returns the absolute value
sqrt – returns the square root
exp – returns the value of the exponential function
log – returns the natural log

```
*Program 11-7 Demonstrating the ABS, SQRT, EXP, and LOG functions - page 197;
 data math;
    input x @@;
    Absolute = abs(x);
    Square = sqrt(x);
    Exponent = exp(x);
    Natural = log(x);
 datalines;
 2 -2 10 100
 ;
 run;
```

This example also illustrates the use of the @@ on the INPUT statement.

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

The “@@” is needed when you don’t instruct SAS as to how many data values it will need to read on the line(s) of data.

Compare this to the example on page 130 which uses a single @ on the INPUT statement.

In that example, only a single ‘@’ is required, because the INPUT statement is contained within the do loop which controls the number of times the input statement is read.

11.8 Computing Some Useful Constants

This section is omitted.

11.9 Generating Random Numbers

This section is omitted.

11.10 Special Functions

There are two functions that are of great importance and are widely used, the INPUT and PUT functions.

INPUT – converts character values to numeric values
PUT – converts numeric values to a character values

The examples for this section also illustrate statements that can be used to keep your data sets tidy.
These statements are:

DROP – drops variables from a data set
KEEP – keeps variables in a data set (dropping all other variables)
RENAME – renames variables

You do not use both DROP and KEEP at the same time.
The rule of thumb is to use the statement that requires less coding.
If you are dropping many variables and keeping a few, use KEEP.
If you are dropping a few variables and keeping many, use DROP.

When coded within a data step the syntax is:

```
DROP variablename1 variablename2...variablename'n';
KEEP variablename1 variablename2...variablename'n';
RENAME oldvariablename1=newvariablename1 oldvariablename2=newvariablename2;
```

Typically these are placed at or near the bottom of your data step code.

All three of these options can be applied to both the incoming data set (on the SET statement) and the data set being created.

The syntax is:

```
data datasetname(drop=variablename1 variablename2...variablename'n'
                 rename=(oldvariablename1=newvariablename1
                         oldvariablename2=newvariablename2));
  set datasetname(keep=variablename1 variablename2...variablename'n');
```

When coded in this way, it is important to understand how and when these options are applied.

The options on the SET statement are applied first since it is the incoming data set.

If variables are renamed, code within the data step must be written using the NEW variable names.

Within the data step code, RENAME and DROP are the last statements executed.

Thus if variables are renamed within the data step, code must be written using the OLD variable names.

```
*Program 11-11 Using the INPUT function to perform a character-to-numeric conversion - page 202;
data nums;
  set learn.chars (rename=
                  (Height = Char_Height
                   Weight = Char_Weight
                   Date   = Char_Date));
 Height = input(Char_Height,8.);
 Weight = input(Char_Weight,8.);
 Date   = input(Char_Date,mmddyy10.);
 drop Char_Height Char_Weight Char_Date;
run;
```

In this example, the data set learn.chars contains 3 character variables called Height, Weight, and Date. This can be verified by viewing the data set in your session or by running PROC CONTENTS. The data values are displayed on page 201.

This examples illustrates how to create a new data set, nums, that will contain these data in 3 numeric variables called Height, Weight, and Date.

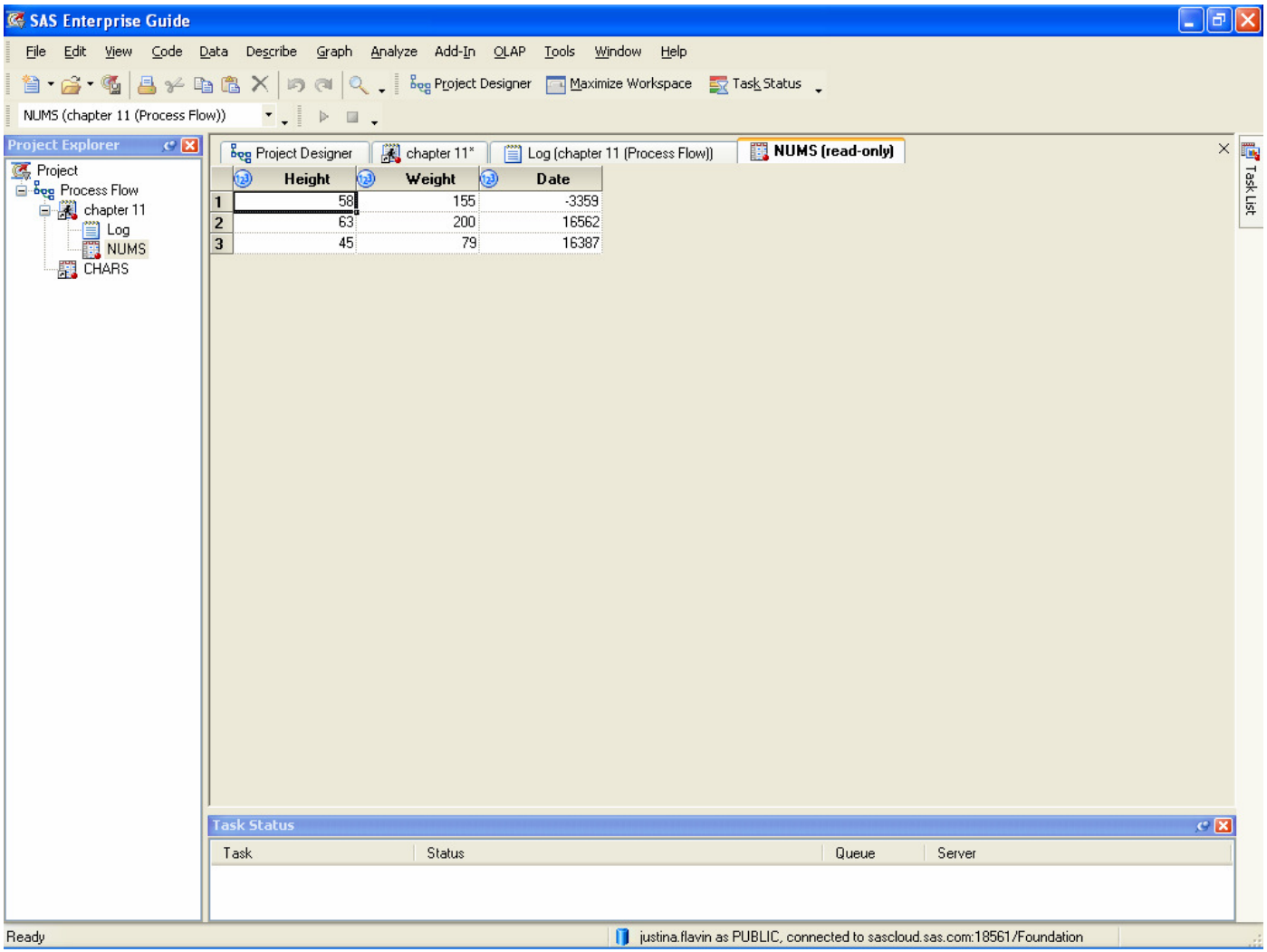
First, the rename statement on the SET statement is applied. This causes the variables to be renamed Char_Height, Char_Weight, and Char_Date. Note: this does not change the variable names in LEARN.CHARS data set.

Next, new numeric variables called Height, Weight, and Date are created using the INPUT function and the appropriate informat.

The character variables Char_Height Char_Weight Char_Date are no longer needed or wanted, so these are deleted from the data set using the DROP statement.

This same technique could be used to covert numeric variables to character variables, using the PUT function instead of the INPUT function. The PUT function is illustrated in the next example.

We now have a data set with the same values stored as numeric variables. Does the Date variable appear to be correct? What can you do to improve its appearance and make it look like a calendar date?



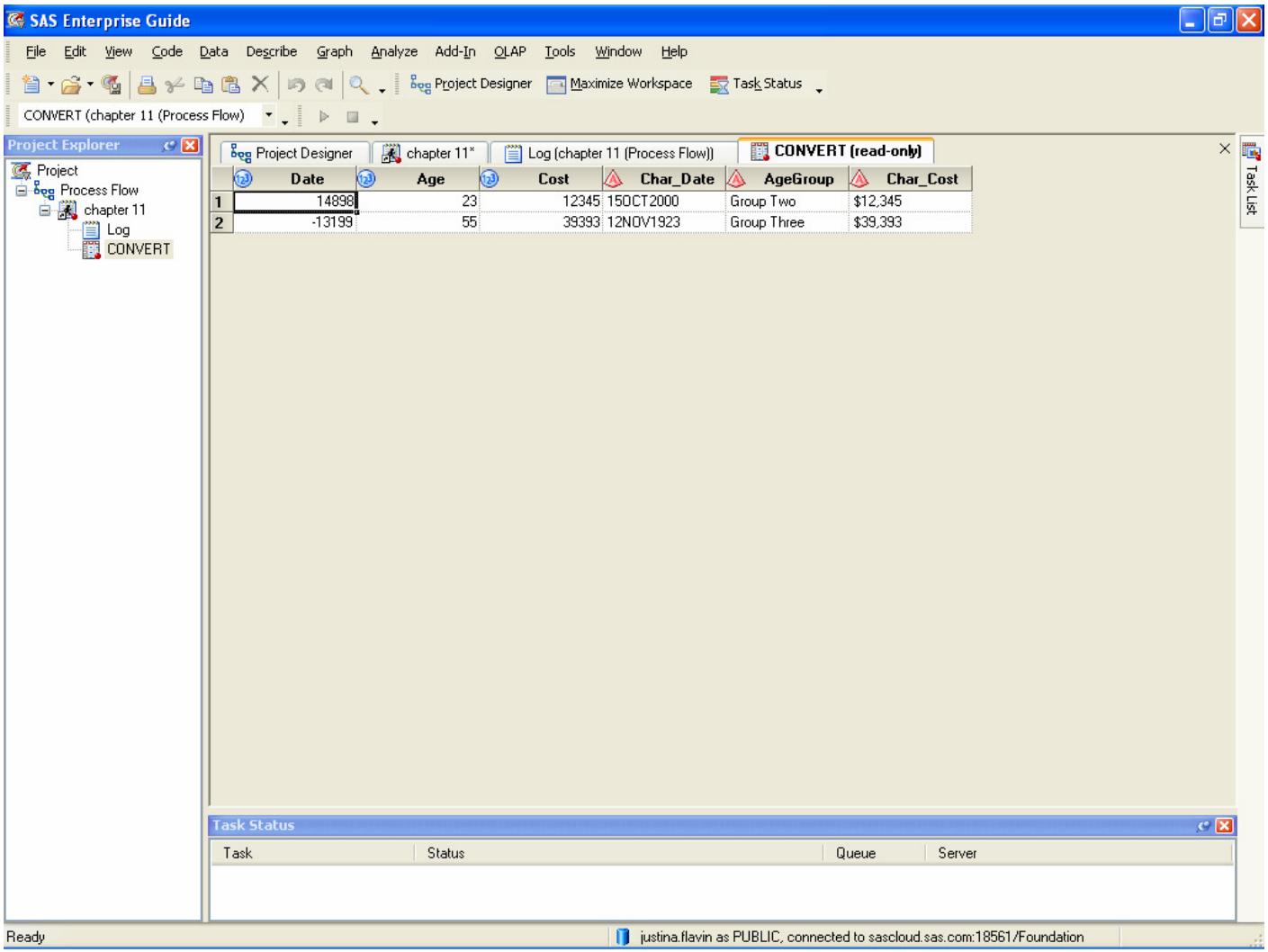
```
*Program 11-12 Demonstrating the PUT function - page 203;
proc format;
   value agefmt low-<20 = 'Group One'
                20-<40  = 'Group Two'
                40-high = 'Group Three';
run;

data convert;
   set learn.numeric;
   Char_Date = put(Date,date9.);
   AgeGroup = put(Age,agefmt.);
   Char_Cost = put(Cost,dollar10.);
   drop Date Cost;
run;
```

This example illustrates using the PUT function to convert numeric variables to character variables.

While an informat is used with the INPUT statement, a format is used with the PUT statement.

A grouping format is created for the variable Age, and this formatted (character string) value is used.

Here is data set CONVERT



11.11 Functions That Return Values from Previous Observations

This section is omitted.