Chapter 14 MODIFYING NUMERIC VALUES WITH FUNCTIONS

* To return the integer portion of a numeric value, use the function **INT**. The syntax is int(*argument*) where argument may be the name of a numeric variable, a constant, or an expression.

Example. The numbers below are truncated by the INT function.

data numbers;

input values;

integers=int(values);

cards;

538.694226

6.24739599

466149.252

-27.5157162

;

proc print noobs;

format values 11.7;

run;

The output is

values integers

538.6942260 538

6.2473960 6

466149.2520 466149

-27.5157162 -27

* To round value to the nearest specified decimal place, use the function **ROUND**. The syntax is round(*argument*, *round-off unit*)

The default round-off unit is one, and the value is rounded to the nearest integer.

Example. The numbers below are rounded off the two decimal places by the function ROUND.

data numbers;

input values;

rounded\_values=round(values,0.01);

cards;

538.694226

6.24739599

466149.252

-27.5157162

;

proc print noobs;

format values 11.7;

run;

The output is

rounded\_

values values

538.6942260 538.69

6.2473960 6.25

466149.2520 466149.25

-27.5157162 -27.52

Chapter 11 CREATING AND MANAGING VARIABLES

ACCUMULATING TOTALS

It is often useful to create a variable that accumulates the values of another variable. The syntax is (in the DATA step)

*variable\_name*+*expression*;

where *variable\_name* is the name of the **accumulator variable**. Its initial value is set to zero by default. Its value is retained from one DATA step execution to the next.

Example. Consider the data set for the exercise example.

| **ID** | **Age** | **ActLevel** | **Gender** | **Payment Due** |
| --- | --- | --- | --- | --- |
| 2810 | 61 | MOD | F | 34.00 |
| 2804 | 38 | HIGH | F | 29.00 |
| 2807 | 42 | LOW | M | 27.50 |
| 2816 | 26 | HIGH | M | 20.25 |
| 2833 | 32 | MOD | F | 15.50 |
| 2823 | 29 | HIGH | M | 23.00 |

The following DATA step computes the total of payments due. The calculations are done iteratively.

data exercise;

input id $ age actlevel $ gender $ payment;

total+payment;

cards;

2810 61 MOD F 34.00

2804 38 HIGH F 29.00

2807 42 LOW M 27.50

2816 26 HIGH M 20.25

2833 32 MOD F 15.50

2823 29 HIGH M 23.00

;

proc print noobs;

run;

The output is

id age actlevel gender payment total

2810 61 MOD F 34.00 34.00

2804 38 HIGH F 29.00 63.00

2807 42 LOW M 27.50 90.50

2816 26 HIGH M 20.25 110.75

2833 32 MOD F 15.50 126.25

2823 29 HIGH M 23.00 149.25

THE RETAIN STATEMENT

The **RETAIN** statement is used to assign to an accumulator variable an **initial value** other than the default value of zero. The syntax is

retain *variable\_name* *initial\_value*;

Example. Suppose in the exercise example, the payment due for another person is $30, so the initial total payment due is set at 30.

data exercise;

input id $ age actlevel $ gender $ payment;

retain Total 30;

total+payment;

cards;

2810 61 MOD F 34.00

2804 38 HIGH F 29.00

2807 42 LOW M 27.50

2816 26 HIGH M 20.25

2833 32 MOD F 15.50

2823 29 HIGH M 23.00

;

proc print noobs;

run;

The output is

id age actlevel gender payment total

2810 61 MOD F 34.00 64.00

2804 38 HIGH F 29.00 93.00

2807 42 LOW M 27.50 120.50

2816 26 HIGH M 20.25 140.75

2833 32 MOD F 15.50 156.25

2823 29 HIGH M 23.00 179.25

THE IF-THEN/ELSE STATEMENT

To perform an action conditionally, use an **IF-THEN/ELSE** statement. The **IF-THEN** statement executes a SAS statement when the condition in the IF clause is true. The **optional ELSE** statement is executed when the IF clause is false. The syntax is

if *expression* then *statement*;

else *statement*;

Any of the following comparison operators may be used in IF-THEN/ELSE statement:

|  |  |
| --- | --- |
| Operator | Name of Operation |
| = or eq | equal to |
| ^= or ne | not equal to |
| > or gt | freater than |
| < or lt | less than |
| >= or ge | greater than or equal to |
| <=or le | less than or equal to |
| in | equal to one of a list |

Example. In the exercise example, a new variable Gender is created using IF-THEN/ELSE statement.

data exercise;

set exercise;

if gender='F' then gender1='Female';

else gender1='Male';

run;

proc print noobs;

run;

The output is

id age actlevel gender payment total gender1

2810 61 MOD F 34.00 64.00 Female

2804 38 HIGH F 29.00 93.00 Female

2807 42 LOW M 27.50 120.50 Male

2816 26 HIGH M 20.25 140.75 Male

2833 32 MOD F 15.50 156.25 Female

2823 29 HIGH M 23.00 179.25 Male

data exercise;

set exercise;

informat exersion $9.;

if gender='F' then gender1='Female';

else gender='Male';

if age>40 then age\_new='senior'; else age\_new='junior';

/\*or age\_new=(age>40);\*/

if actlevel='MOD' then exersion='moderate';

else do;

if actlevel='HIGH' then exersion='intensive';

else exersion='low';end;

run;

proc print noobs;

format exersion $9.;

run;

THE LENGTH STATEMENT

The **LENGTH** statement specifies a length (the number of bytes) of a variable. The syntax is

length *variable\_name* <$> *length*;

where the optional $ indicates that the variable is a character variable.

Example. Suppose in the exercise example the variable Gender is created by using the following code:

data exercise;

set exercise;

if gender='M' then gender1='Male';

else gender1='Female';

run;

proc print noobs;

run;

The output is

id age actlevel gender payment gender1

2810 61 MOD F 34.00 Fema

2804 38 HIGH F 29.00 Fema

2807 42 LOW M 27.50 Male

2816 26 HIGH M 20.25 Male

2833 32 MOD F 15.50 Fema

2823 29 HIGH M 23.00 Male

Note that the value ‘Female’ is truncated to match the value ‘Male’ in length. SAS allocates as many bytes of storage space as there are characters in the **first encounter** for that variable. The first value in this case is ‘Male’ in the IF-THEN statement, which specifies a four-character value (‘Male’). To avoid truncation, set the length of the variable gender1 to the longest value (‘Female’), which has six characters. It can be done by adding the LENGTH statement.

data exercise;

set exercise;

length gender1 $ 6;

if gender='M' then gender1='Male';

else gender1='Female';

run;

proc print noobs;

run;

The output is

id age actlevel gender payment gender1

2810 61 MOD F 34.00 Female

2804 38 HIGH F 29.00 Female

2807 42 LOW M 27.50 Male

2816 26 HIGH M 20.25 Male

2833 32 MOD F 15.50 Female

2823 29 HIGH M 23.00 Male

THE DELETE STATEMENT

The IF-THEN statement may be used in conjunction with the **DELETE** statement to omit unwanted observations. The syntax is

if *expression* then delete;

Example. In the exercise example, all females are deleted from the data set by the following code.

data exercise;

set exercise;

if gender='F' then delete;

run;

proc print noobs;

run;

The output is

id age actlevel gender payment

2807 42 LOW M 27.50

2816 26 HIGH M 20.25

2823 29 HIGH M 23.00

THE SELECT STATEMENT

The **SELECT** statement may be used in the DATA step to assign values conditionally to a variable. This is an alternative to using a series of the IF-THEN/ELSE statements. The syntax is

select(*variable\_name*);

when (*value1*) *new\_variable*=*new\_value1*;

when (*value2, value3, …*) *new\_variable*=*new\_value2*;

<otherwise *new\_variable*=*new\_value*;>

end;

* The values *value1*, *value2*, etc. are the values assumed by *variable\_name*.
* The use of *otherwise* is optional.
* The statement ends with *end;*

Example. In the exercise example, the following code creates a new variable Level that contains description of activity levels.

data exercise;

set exercise;

length level $ 23;

select(actlevel);

when('HIGH') level='High Activity Level';

when('MOD') level='Moderate Activity Level';

otherwise level='Low Activity Level';

end;

run;

proc print noobs;

run;

The output is

id age actlevel gender payment level

2810 61 MOD F 34.00 Moderate Activity Level

2804 38 HIGH F 29.00 High Activity Level

2807 42 LOW M 27.50 Low Activity Level

2816 26 HIGH M 20.25 High Activity Level

2833 32 MOD F 15.50 Moderate Activity Level

2823 29 HIGH M 23.00 High Activity Level

THE DO STATEMENT

The DO statement may be used in conjunction with the IF-THEN/ELSE and SELECT statements to execute several statements when a condition is true. The syntax is

do;

*SAS statements*;

end;

Example. In the exercise example, the following program creates a new variable Level and calculates the total payments due for each value of this variable.

data exercise;

set exercise;

length level $ 23;

select(actlevel);

when('HIGH')

do;

level='High Activity Level';

total\_high+payment;

end;

when('MOD')

do;

level='Moderate Activity Level';

total\_mod+payment;

end;

when('LOW')

do;

level='Low Activity Level';

total\_low+payment;

end;

end;

run;

proc print noobs;

run;

The output is

total\_ total\_ total\_

id age actlevel gender payment level high mod low

2810 61 MOD F 34.00 Moderate Activity Level 0.00 68.0 0.0

2804 38 HIGH F 29.00 High Activity Level 58.00 34.0 0.0

2807 42 LOW M 27.50 Low Activity Level 29.00 34.0 55.0

2816 26 HIGH M 20.25 High Activity Level 69.50 34.0 27.5

2833 32 MOD F 15.50 Moderate Activity Level 49.25 65.0 27.5

2823 29 HIGH M 23.00 High Activity Level 95.25 49.5 27.5