Using the Imported Data in a DATA Step

Naming the Data Set with the DATA Statement

The DATA statement indicates the beginning of the DATA step and names the SAS data set to be created.

Syntax, DATA statement:

DATA *SAS-data-set-1* <...*SAS-data-set-n*>;

SAS-data-set names (in the format *libref.filename*) the data set or data sets to be created.

Remember that a permanent SAS data set name is a two-level name. For example, the two-level name Clinic.Admit specifies that the data set Admit is stored in the permanent SAS library to which the libref Clinic has been assigned.

Specifying the Imported Data with the SET Statement

The SET statement specifies the SAS data set that you want to use as input data for your DATA step. When you import your external data using PROC IMPORT, you specify the name of the output data set using the OUT= option. Use the libref and data set name that you specified using the OUT= option as the SAS data set value for the SET statement.

SET Statement Syntax

Syntax, DATA step for reading a single data set:

DATA *SAS-data-set*;
        SET *SAS-data-set*;
    <...more SAS statements...>
RUN;

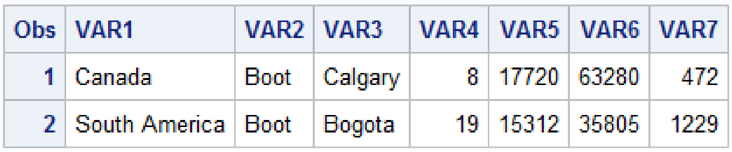
- *SAS-data-set* in the DATA statement is the name of the SAS data set to be created.
- *SAS-data-set* in the SET statement is the name of the SAS data set to be read.

Example: Using the SET Statement to Specify Imported Data

In this example, the DATA statement tells SAS to name the new data set, Boots, and store it in the temporary library Work. The SET statement in the DATA step specifies the output data set from the IMPORT procedure. You can use several statements in the DATA step to subset your data as needed. In this example, the WHERE statement is used with VAR1 to include only the observations where VAR1 is either South America or Canada.

```
proc import datafile="C:\certdata\boot.csv"
  out=shoes
  dbms=csv
  replace;
  getnames=no;
run;
data boots;
  set shoes;
  where var1='South America' OR var1='Canada';
run;
```

Output 4.7 *Results from the DATA Step Using the SET Statement*



Obs	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7
1	Canada	Boot	Calgary	8	17720	63280	472
2	South America	Boot	Bogota	19	15312	35805	1229

Reading a Single SAS Data Set to Create Another

Example: Reading a SAS Data Set

The data set Cert.Admit contains health information about patients in a clinic, their activity level, height, and weight. Suppose you want to create a subset of the data. Specifically, you want to create a small data set containing data about all the men in the group who are older than 50.

To create the data set, you must first reference the library in which Cert.Admit is stored. Then you must specify the name of the library in which you want to store the Males data set. Finally, you add statements to the DATA step to read your data and create a new data set.

The DATA step below reads all observations and variables from the existing data set Cert.Admit into the new data set Males. The DATA statement creates the permanent

SAS data set Males, which is stored in the SAS library Men50. The SET statement reads the permanent SAS data set Cert.Admit and subsets the data using a WHERE statement. The new data set, Males, contains all males in Cert.Admit who are older than 50.

```
libname cert 'C:\Users\Student1\cert\';
libname Men50 'C:\Users\Student1\cert\Men50';
data Men50.males;
  set cert.admit;
  where sex='M' and age>50;
run;
```

When you submit this DATA step, the following messages appear in the log, confirming that the new data set was created:

Log 4.3 *SAS Log Output*

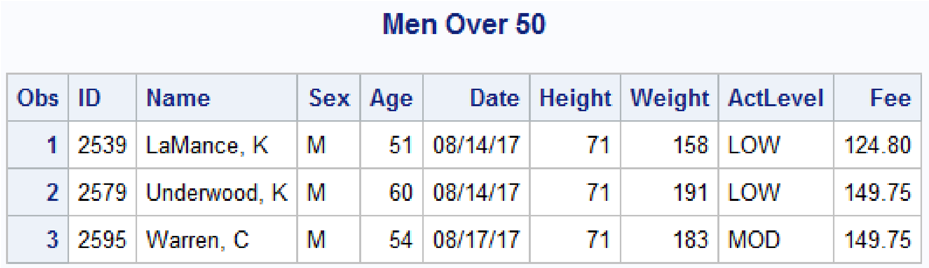
```
69205  data Men50.males;
69206    set cert.admit;
69207    where sex='M' and age>50;
69208  run;

NOTE: There were 3 observations read from the data set
      CERT.ADMIT.
      WHERE (sex='M') and (age>50);
NOTE: The data set MEN50.MALES has 3 observations and 9
      variables.
```

You can add a PROC PRINT statement to this same example to see the output of Men50.Males.

```
proc print data=Men50.males;
   title 'Men Over 50';
run;
```

Output 4.8 *PROC PRINT Output for the Data Set Males*



Men Over 50

Obs	ID	Name	Sex	Age	Date	Height	Weight	ActLevel	Fee
1	2539	LaMance, K	M	51	08/14/17	71	158	LOW	124.80
2	2579	Underwood, K	M	60	08/14/17	71	191	LOW	149.75
3	2595	Warren, C	M	54	08/17/17	71	183	MOD	149.75

Specifying DROP= and KEEP= Data Set Options

You can specify the DROP= and KEEP= data set options anywhere you name a SAS data set. You can specify DROP= and KEEP= in either the DATA statement or the SET statement. It depends on whether you want to drop variables from either the output data set or the source data set:

- If you never reference certain variables and you do not want them to appear in the new data set, use a DROP= option in the SET statement.

  In the DATA step shown below, the DROP= or KEEP= option in the SET statement prevents the variables *Triglyc* and *Uric* from being read. These variables do not appear in the Cert.Drug1h data set and are not available to be used by the DATA step.

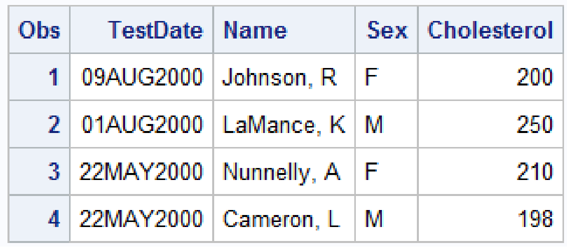
- If you do need to reference a variable in the original data set (in a subsetting IF statement, for example), you can specify the variable in the DROP= or KEEP= option in the DATA statement. Otherwise, the statement that references the variable uses a missing value for that variable.

  This DATA step uses the variable Placebo to select observations. To drop Placebo from the new data set, the DROP= option must appear in the DATA statement.

  When used in the DATA statement, the DROP= option simply drops the variables from the new data set. However, they are still read from the original data set and are available within the DATA step.

```
data cert.drug1h(drop=placebo);
  set cert.cltrials(drop=triglyc uric);
  if placebo='YES';
run;
proc print data=cert.drug1h;
run;
```

Output 4.9 *PROC PRINT Output of Cert.Drug1h*



Obs	TestDate	Name	Sex	Cholesterol
1	09AUG2000	Johnson, R	F	200
2	01AUG2000	LaMance, K	M	250
3	22MAY2000	Nunnelly, A	F	210
4	22MAY2000	Cameron, L	M	198

Creating Variables

Assignment Statements

Use an assignment statement in any DATA step in order to modify existing values or create new variables.

Syntax, assignment statement:

variable=*expression*;

- *variable* names a new or existing variable
- *expression* is any valid SAS expression

Tip: The assignment statement is one of the few SAS statements that do not begin with a keyword.

For example, here is an assignment statement that assigns the character value **`Toby Witherspoon`** to the variable Name:

```
Name='Toby Witherspoon';
```

SAS Expressions

You use SAS expressions in assignment statements and many other SAS programming statements to do the following:

- transform variables
- create new variables
- conditionally process variables
- calculate new values
- assign new values

An expression is a sequence of operands and operators that form a set of instructions.

- Operands are variable names or constants. They can be numeric, character, or both.
- Operators are special-character operators, grouping parentheses, or functions.

Using Operators in SAS Expressions

Use the following arithmetic operators to perform a calculation.

Table 9.1 *Arithmetic Operators*

Operator	Action	Example	Priority
-	negative prefix	`negative=-x;`	1
**	exponentiation	`raise=x**y;`	1
*	multiplication	`mult=x*y;`	2
/	division	`divide=x/y;`	2
+	addition	`sum=x+y;`	3
-	subtraction	`diff=x-y;`	3

The order of operation is determined by the following conditions:

- Operations of priority 1 are performed before operations of priority 2, and so on.
- Consecutive operations that have the same priority are performed in this order:
  - from right to left within priority 1
  - from left to right within priority 2 and 3
- You can use parentheses to control the order of operations.

Note: When a value that is used with an arithmetic operator is missing, the result of the expression is missing. The assignment statement assigns a missing value to a variable if the result of the expression is missing.

Use the following comparison operators to express a condition.

Table 9.2 *Comparison Operators*

Operator	Meaning	Example
= or eq	equal to	`name='Jones, C.'`
^= or ne	not equal to	`temp ne 212`
> or gt	greater than	`income>20000`

< or lt	less than	`x=5000`
		`x<8000`

>= or ge	greater than or equal to	`x=5000`
		`x>=2000`

<= or le	less than or equal to	`pulse le 85`

Use logical operators to link a sequence of expressions into compound expressions.

Table 9.3 *Logical Operators*

Operator, symbol	Description
AND or &	and, both. If both expressions are true, then the compound expression is true.
OR or |	or, either. If either expression is true, then the compound expression is true.

Note: In SAS, any numeric value other than 0 or missing is true, and a value of 0 or missing is false. Therefore, a numeric variable or expression can stand alone in a condition.

- 0 = False
- . = False
- 1 = True

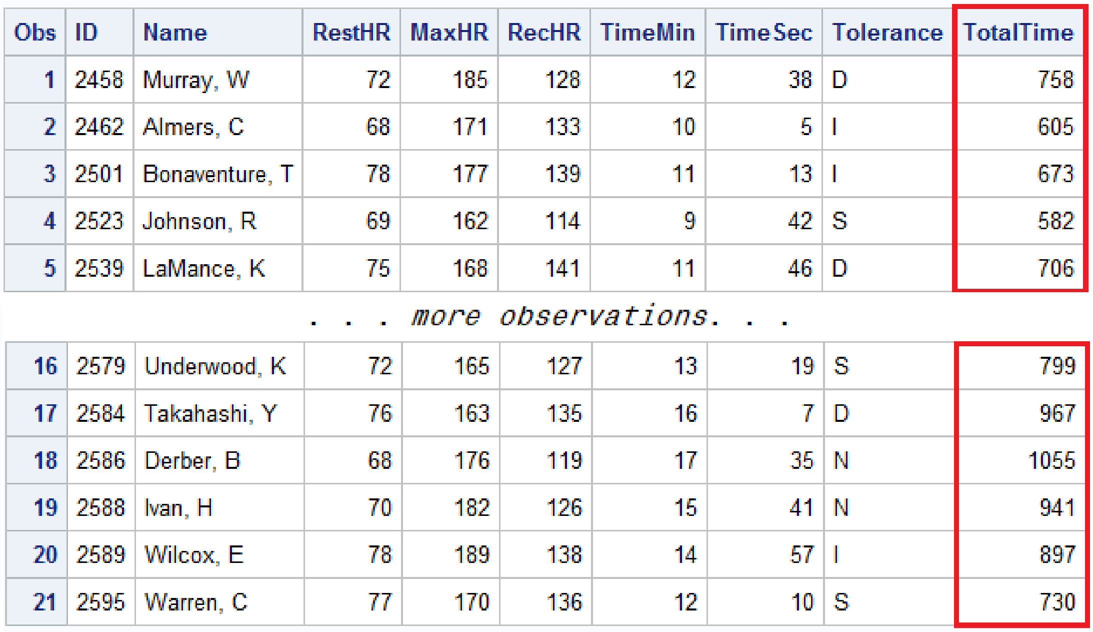
Examples: Assign Variables

Example 1: Create a New Variable

The assignment statement in the DATA step below creates a new variable, TotalTime, by multiplying the values of TimeMin by 60 and then adding the values of TimeSec.

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
run;
proc print data=work.stresstest;
run;
```

Output 9.1 *Assignment Statement Output (partial output)*



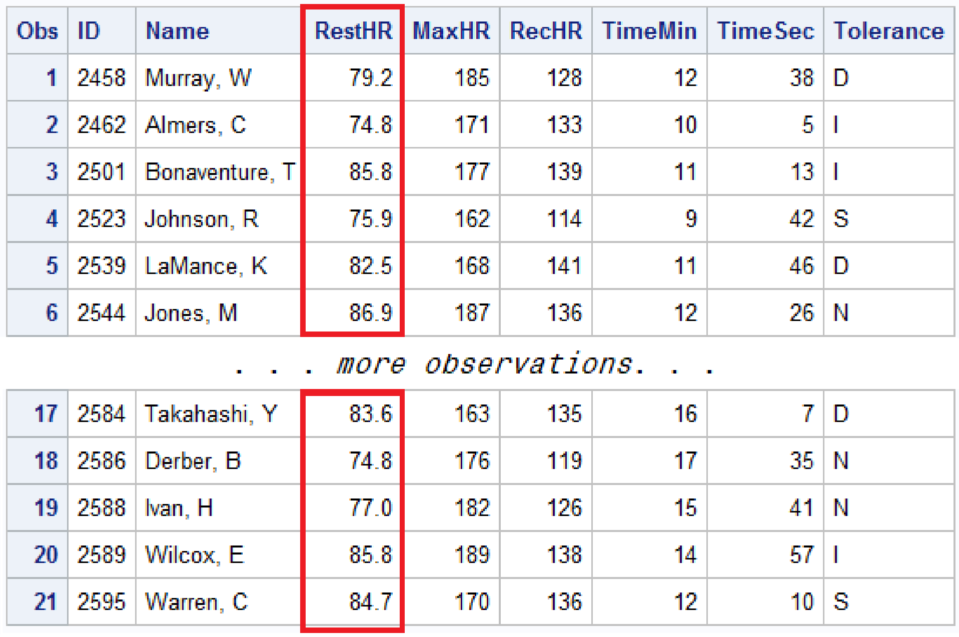
Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
1	2458	Murray, W	72	185	128	12	38	D	758
2	2462	Almers, C	68	171	133	10	5	I	605
3	2501	Bonaventure, T	78	177	139	11	13	I	673
4	2523	Johnson, R	69	162	114	9	42	S	582
5	2539	LaMance, K	75	168	141	11	46	D	706
. . . more observations. . .									
16	2579	Underwood, K	72	165	127	13	19	S	799
17	2584	Takahashi, Y	76	163	135	16	7	D	967
18	2586	Derber, B	68	176	119	17	35	N	1055
19	2588	Ivan, H	70	182	126	15	41	N	941
20	2589	Wilcox, E	78	189	138	14	57	I	897
21	2595	Warren, C	77	170	136	12	10	S	730

Example 2: Re-evaluating Variables

In the following example, the assignment statement contains the variable RestHR, which appears on both sides of the equal sign. This assignment statement evaluates each observation to redefine each RestHR observation as 10% higher. When a variable name appears on both sides of the equal sign, the original value on the right side is used to evaluate the expression. The result is assigned to the variable on the left side of the equal sign.

```
data work.stresstest;
  set cert.tests;
  resthr=resthr+(resthr*.10);
run;
proc print data=work.stresstest;
run;
```

Output 9.2 *PROC PRINT Output of Work.StressTest (partial output)*



Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance
1	2458	Murray, W	79.2	185	128	12	38	D
2	2462	Almers, C	74.8	171	133	10	5	I
3	2501	Bonaventure, T	85.8	177	139	11	13	I
4	2523	Johnson, R	75.9	162	114	9	42	S
5	2539	LaMance, K	82.5	168	141	11	46	D
6	2544	Jones, M	86.9	187	136	12	26	N

. . . more observations. . .

Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance
17	2584	Takahashi, Y	83.6	163	135	16	7	D
18	2586	Derber, B	74.8	176	119	17	35	N
19	2588	Ivan, H	77.0	182	126	15	41	N
20	2589	Wilcox, E	85.8	189	138	14	57	I
21	2595	Warren, C	84.7	170	136	12	10	S

Date Constants

You can assign date values to variables in assignment statements by using date constants. SAS converts a date constant to a SAS date. To represent a constant in SAS date form, specify the date as *'ddmmmyy'* or *'ddmmmyyyy'*, immediately followed by a D.

Syntax, date constant:

'*ddmmmyy*'*d*

or

'*ddmmmyy*'*d*

- *dd* is a one- or two-digit value for the day.
- *mmm* is a three-letter abbreviation for the month (JAN, FEB, and so on).
- *yy* or *yyyy* is a two- or four-digit value for the year, respectively.

Tip: Be sure to enclose the date in quotation marks.

TIP You can also use SAS time constants and SAS datetime constants in assignment statements.

```
Time='9:25't;
DateTime='18jan2018:9:27:05'dt;
```

Example: Assignment Statements and Date Values

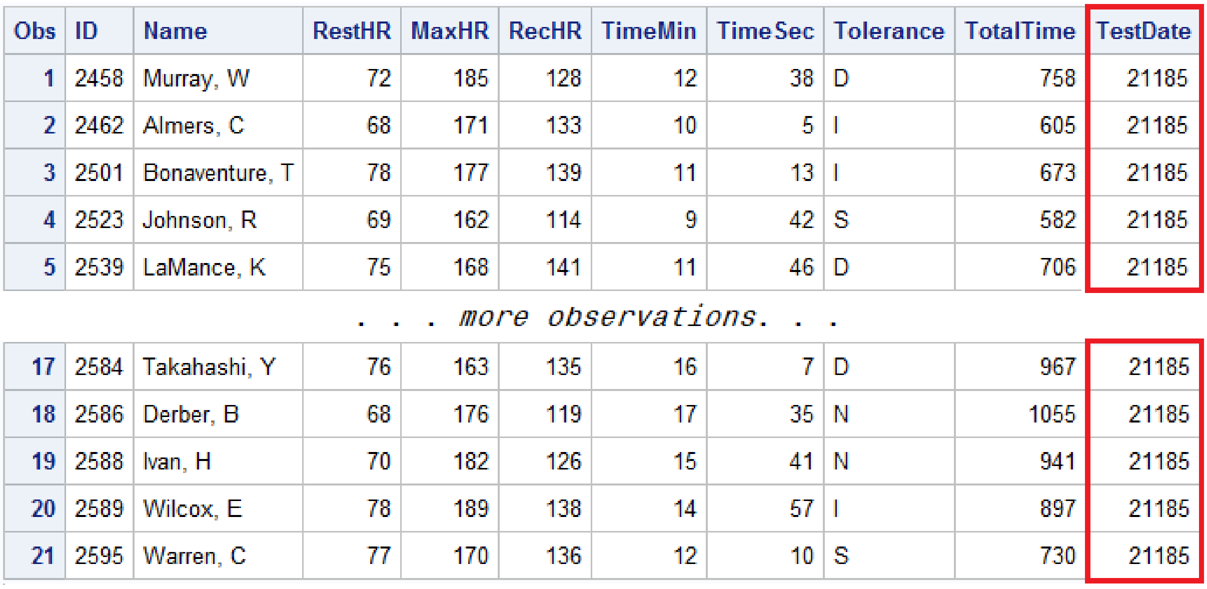
In the following program, the second assignment statement assigns a date value to the variable TestDate.

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  TestDate='01jan2015'd;
```

```
run;
proc print data=work.stresstest;
run;
```

Notice how the values for TestDate in the PROC PRINT output are displayed as SAS date values.

Output 9.3 *PROC PRINT Output of Work.StressTest with SAS Date Values (partial output)*



Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime	TestDate
1	2458	Murray, W	72	185	128	12	38	D	758	21185
2	2462	Almers, C	68	171	133	10	5	I	605	21185
3	2501	Bonaventure, T	78	177	139	11	13	I	673	21185
4	2523	Johnson, R	69	162	114	9	42	S	582	21185
5	2539	LaMance, K	75	168	141	11	46	D	706	21185

. . . more observations. . .

Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime	TestDate
17	2584	Takahashi, Y	76	163	135	16	7	D	967	21185
18	2586	Derber, B	68	176	119	17	35	N	1055	21185
19	2588	Ivan, H	70	182	126	15	41	N	941	21185
20	2589	Wilcox, E	78	189	138	14	57	I	897	21185
21	2595	Warren, C	77	170	136	12	10	S	730	21185

You can use a FORMAT statement in the PROC PRINT step to modify the TestDate values and change them to another format. To apply formats to your output, see Lesson 12, “SAS Formats and Informats,” on page 205.

Modifying Variables

Selected Useful Statements

Here are examples of statements that accomplish specific data-manipulation tasks.

Table 9.4 *Manipulating Data Using the DATA Step*

Task	Example Code

Subset data	`if resthr<70 then delete;`
	`if tolerance='D';`

Drop unwanted variables	`drop timemin timesec;`
Create or modify a variable	`TotalTime=(timemin*60)+timesec;`
Initialize and retain a variable	`retain SumSec 5400;`
Accumulate totals	`sumsec+totaltime;`

Task	Example Code
Specify a variable's length	`length TestLength $ 6;`

Execute statements conditionally	`if totaltime>800 then TestLength='Long';`
	`else if 750<=totaltime<=800`
	`  then TestLength='Normal';`
	`    else if totaltime<750`
	`  then TestLength='Short';`

The following topics discuss these tasks.

Accumulating Totals

To add the result of an expression to an accumulator variable, you can use a sum statement in your DATA step.

Syntax, sum statement:

variable+*expression*;

- *variable* specifies the name of the accumulator variable. This variable must be numeric. The variable is automatically set to 0 before the first observation is read. The variable's value is retained from one DATA step execution to the next.
- *expression* is any valid SAS expression.

Note: If the expression produces a missing value, the sum statement ignores it.

The sum statement is one of the few SAS statements that do not begin with a keyword.

The sum statement adds the result of the expression that is on the right side of the plus sign (+) to the numeric variable that is on the left side of the plus sign. The value of the accumulator variable is initialized to 0 instead of missing before the first iteration of the DATA step. Subsequently, the variable’s value is retained from one iteration to the next.

Example: Accumulating Totals

To find the total number of elapsed seconds in treadmill stress tests, you need the variable SumSec, whose value begins at 0 and increases by the amount of the total seconds in each observation. To calculate the total number of elapsed seconds in treadmill stress tests, use the sum statement shown below:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  SumSec+totaltime;
run;
```

The value of the variable on the left side of the plus sign, SumSec, begins at 0 and increases by the value of TotalTime with each observation.

SumSec	=	TotalTime	+	Previous total
0				

SumSec	=	TotalTime	+	Previous total
758	=	758	+	0
1363	=	605	+	758
2036	=	673	+	1363
2618	=	582	+	2036
3324	=	706	+	2618

Initializing Sum Variables

In the previous example, the sum variable SumSec was initialized to 0 before the first observation was read. However, you can initialize SumSec to a different number than 0.

Use the RETAIN statement to assign an initial value, other than 0, to an accumulator variable in a sum statement.

The RETAIN statement has several purposes:

- It assigns an initial value to a retained variable.
- It prevents variables from being initialized each time the DATA step executes.

Syntax, RETAIN statement for initializing sum variables:

RETAIN *variable* <*initial-value*>;

- *variable* is a variable whose values you want to retain.
- *initial-value* specifies an initial value (numeric or character) for the preceding variable.

Note: The following statements are true about the RETAIN statement:

- It is a compile-time-only statement that creates variables if they do not already exist.
- It initializes the retained variable to missing before the first execution of the DATA step if you do not supply an initial value.
- It has no effect on variables that are read with SET, MERGE, or UPDATE statements.

Example: RETAIN Statement

Suppose you want to add 5400 seconds (the accumulated total seconds from a previous treadmill stress test) to the variable SumSec in the StressTest data set when you create the data set. To initialize SumSec with the value 5400, use the RETAIN statement shown below. Now the value of SumSec begins at 5400 and increases by the value of TotalTime with each observation.

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
run;
proc print data=work.stresstest;
run;
```

SumSec	=	TotalTime	+	Previous Total
5400				
6158	=	758	+	5400
6763	=	605	+	6158
7436	=	673	+	6763
8018	=	582	+	7436
8724	=	706	+	8018

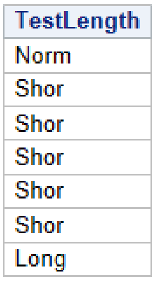
Specifying Lengths for Variables

Avoiding Truncated Variable Values

During the compilation phase, use an assignment statement to create a new character variable. SAS allocates as many bytes of storage space as there are characters in the first value that it encounters for that variable.

In the following figure, the variable TestLength has a length of four bytes. The word Short is truncated because the word Norm uses four bytes.

Figure 9.1 *Truncated Variable Values (partial output)*



TestLength
Norm
Shor
Shor
Shor
Shor
Shor
Long

When you assign a character constant as the value of the new variable, use the LENGTH statement to specify a length to avoid truncation of your values.

Syntax, LENGTH statement:

LENGTH *variable(s)* <$> *length***;**

- *variable(s)* names the variable or variables to be assigned a length.
- $ is specified if the variable is a character variable.
- *length* is an integer that specifies the length of the variable.

Here is a variable list in which all three variables are assigned a length of $200.

```
length Address1 Address2 Address3 $200;
```

Example: LENGTH Statement

Within the program, a LENGTH statement is included to assign a length to accommodate the longest value of the variable TestLength. The longest value is **Normal**, which has six characters. Because TestLength is a character variable, you must follow the variable name with a dollar sign ($).

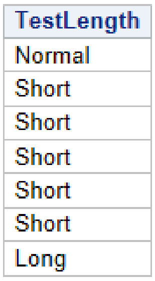
Make sure the LENGTH statement appears before any other reference to the variable in the DATA step.

```
data stress;
  set cert.stress;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $ 6;
  if totaltime>800 then testlength='Long';
  else if 750<=totaltime<=800 then testlength='Normal';
  else if totaltime<750 then TestLength='Short';
run;
```

Note: If the variable has been created by another statement, then a later use of the LENGTH statement does not change its length.

Now that the LENGTH statement has been added to the program, the values of TestLength are no longer truncated.

Figure 9.2 *Variable Values That Are Not Truncated (partial output)*



TestLength
Normal
Short
Short
Short
Short
Short
Long

Subsetting Data

Using a Subsetting IF Statement

The subsetting IF statement causes the DATA step to continue processing only those observations that meet the condition of the expression specified in the IF statement. The resulting SAS data set or data sets contain a subset of the original external file or SAS data set.

Syntax, subsetting IF statement:

IF *expression***;**

expression is any valid SAS expression.

- If the expression is true, the DATA step continues to process that observation.
- If the expression is false, no further statements are processed for that observation, and control returns to the top of the DATA step.

Example: Subsetting IF Statement

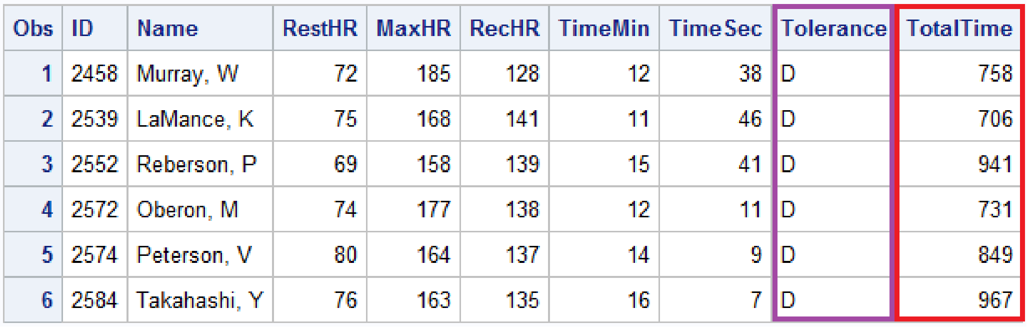
The subsetting IF statement below selects only observations whose values for Tolerance are **D**. It is positioned in the DATA step for efficiency: other statements do not need to process unwanted observations.

```
data work.stresstest;
  set cert.tests;
  if tolerance='D';
    TotalTime=(timemin*60)+timesec;
run;
proc print data=work.stresstest;
run;
```

Because Tolerance is a character variable, the value **D** must be enclosed in quotation marks, and it must be the same case as in the data set.

Notice that, in the output below, only the values where Tolerance contains the value of **D** are displayed and TotalTime was calculated.

Output 9.4 *Subsetted Data of Work.StressTest*



Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
1	2458	Murray, W	72	185	128	12	38	D	758
2	2539	LaMance, K	75	168	141	11	46	D	706
3	2552	Reberson, P	69	158	139	15	41	D	941
4	2572	Oberon, M	74	177	138	12	11	D	731
5	2574	Peterson, V	80	164	137	14	9	D	849
6	2584	Takahashi, Y	76	163	135	16	7	D	967

Categorizing Values

Suppose you want to create a variable that categorizes the length of time that a subject spends on the treadmill during a stress test. This new variable, TestLength, is based on the value of the existing variable TotalTime. The value of TestLength is assigned conditionally:

Value for TotalTime	Resulting Value for TestLength
greater than 800	Long
750 - 800	Normal

Value for TotalTime	Resulting Value for TestLength
less than 750	Short

To perform an action conditionally, use an IF-THEN statement. The IF-THEN statement executes a SAS statement when the condition in the IF clause is true.

Syntax, IF-THEN statement:

IF *expression* **THEN** *statement***;**

- *expression* is any valid SAS expression.
- *statement* is any executable SAS statement.

Example: IF-THEN Statement

To assign the value **`Long`** to the variable TestLength when the value of TotalTime is greater than 800, add the following IF-THEN statement to your DATA step:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  if totaltime>800 then TestLength='Long';
run;
```

SAS executes the assignment statement only when the condition (TotalTime>800) is true. If the condition is false, the value of TestLength is missing.

Examples: Logical Operators

The following examples use IF-THEN statements with logical operators:

- Use the AND operator to execute the THEN statement if both expressions that are linked by AND are true.

  ```
  if status='OK' and type=3
      then Count+1;
  if (age^=agecheck | time^=3)
      & error=1 then Test=1;
  ```

- Use the OR operator to execute the THEN statement if either expression that is linked by OR is true.

  ```
  if (age^=agecheck | time^=3)
      & error=1 then Test=1;
  if status='S' or cond='E'
      then Control='Stop';
  ```

- Use the NOT operator with other operators to reverse the logic of a comparison.

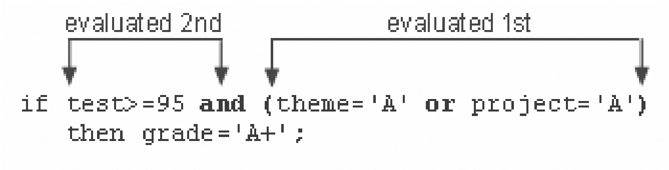
  ```
  if not(loghours<7500)
      then Schedule='Quarterly';
  if region not in ('NE','SE')
      then Bonus=200;
  ```

- Character values must be specified in the same case in which they appear in the data set and must be enclosed in quotation marks.

  ```
  if status='OK' and type=3
      then Count+1;
  if status='S' or cond='E'
      then Control='Stop';
  if not(loghours<7500)
      then Schedule='Quarterly';
  if region not in ('NE','SE')
      then Bonus=200;
  ```

Logical comparisons that are enclosed in parentheses are evaluated as true or false before they are compared to other expressions. In the example below, the OR comparison in parenthesis is evaluated before the first expression and the AND operator are evaluated.

Figure 9.3 Example of a Logical Comparison



```
if test>=95 and (theme='A' or project='A')
   then grade='A+';
```

Therefore, be careful when using the OR operator with a series of comparisons. Remember that only one comparison in a series of OR comparisons must be true to make a condition true, and any nonzero, not missing constant is always evaluated as true. Therefore, the following subsetting IF statement is always true:

```
if x=1 or 2;
```

SAS first evaluates x=1, and the result can be either true or false. However, since the 2 is evaluated as nonzero and not missing (true), the entire expression is true. In this statement, however, the condition is not necessarily true because either comparison can be evaluated as true or false:

```
if x=1 or x=2;
```

Note: Both sides of the OR must contain complete expressions.

Providing an Alternative Action

Suppose you want to assign a value to TestLength based on the other possible values of TotalTime. One way to do this is to add IF-THEN statements for the other two conditions.

```
if totaltime>800 then TestLength='Long';
if 750<=totaltime<=800 then TestLength='Normal';
if totaltime<750 then TestLength='Short';
```

However, when the DATA step executes, each IF statement is evaluated in order, even if the first condition is true. This wastes system resources and slows the processing of your program.

Instead of using a series of IF-THEN statements, you can use the ELSE statement to specify an alternative action to be performed when the condition in an IF-THEN statement is false. As shown below, you can write multiple IF-THEN/ELSE statements to specify a series of mutually exclusive conditions.

```
if totaltime>800 then TestLength='Long';
else if 750<=totaltime<=800 then TestLength='Normal';
else if totaltime<750 then TestLength='Short';
```

The ELSE statement must immediately follow the IF-THEN statement in your program. An ELSE statement executes only if the previous IF-THEN/ELSE statement is false.

Syntax, ELSE statement:

ELSE *statement***;**

statement is any executable SAS statement, including another IF-THEN statement.

To assign a value to TestLength when the condition in your IF-THEN statement is false, you can add the ELSE statement to your DATA step:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
      else if 750<=totaltime<=800 then TestLength='Normal';
      else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

For greater efficiency, construct your IF-THEN/ELSE statements with conditions of decreasing probability.

TIP You can use PUT statements to test your conditional logic.

```
if totaltime>800 then TestLength='Long';
   else if 750<=totaltime<=800 then TestLength='Normal';
      else put 'NOTE: Check this Length: ' totaltime=;
run;
```

Deleting Unwanted Observations

You can specify any executable SAS statement in an IF-THEN statement. For example, you can use an IF-THEN statement with a DELETE statement to determine which observations to omit as you read data.

Syntax, DELETE statement:

DELETE;

To conditionally execute a DELETE statement, use the following syntax for an IF statement:

IF *expression* **THEN DELETE;**

The expression is evaluated as follows:

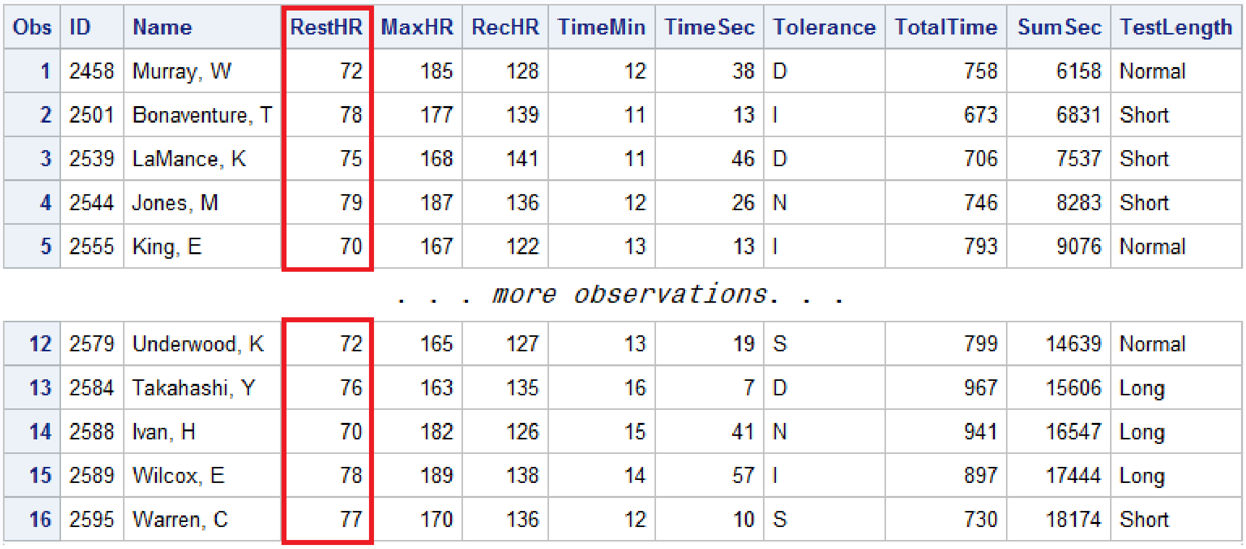
- If it is true, execution stops for that observation. The DELETE statement deletes the observation from the output data set, and control returns to the top of the DATA step.
- If it is false, the DELETE statement does not execute, and processing continues with the next statement in the DATA step.

Example: IF-THEN and DELETE Statements

In the following example, the IF-THEN and DELETE statements omit any observations whose values for RestHR are below 70.

```
data work.stresstest;
  set cert.tests;
  if resthr<70 then delete;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

Output 9.5 *Values for RestHR Less Than 70 Are Not in the Output (partial output)*



Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime	SumSec	TestLength
1	2458	Murray, W	72	185	128	12	38	D	758	6158	Normal
2	2501	Bonaventure, T	78	177	139	11	13	I	673	6831	Short
3	2539	LaMance, K	75	168	141	11	46	D	706	7537	Short
4	2544	Jones, M	79	187	136	12	26	N	746	8283	Short
5	2555	King, E	70	167	122	13	13	I	793	9076	Normal

. . . *more observations.* . . .

Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime	SumSec	TestLength
12	2579	Underwood, K	72	165	127	13	19	S	799	14639	Normal
13	2584	Takahashi, Y	76	163	135	16	7	D	967	15606	Long
14	2588	Ivan, H	70	182	126	15	41	N	941	16547	Long
15	2589	Wilcox, E	78	189	138	14	57	I	897	17444	Long
16	2595	Warren, C	77	170	136	12	10	S	730	18174	Short

Selecting Variables

You might want to read and process variables that you do not want to keep in your output data set. In this case, use the DROP= and KEEP= data set options to specify the variables to drop or keep.

Use the KEEP= option instead of the DROP= option if more variables are dropped than kept.

Syntax, DROP=, and KEEP= data set options:

(DROP=*variable(s)***)**

(KEEP=*variable(s)***)**

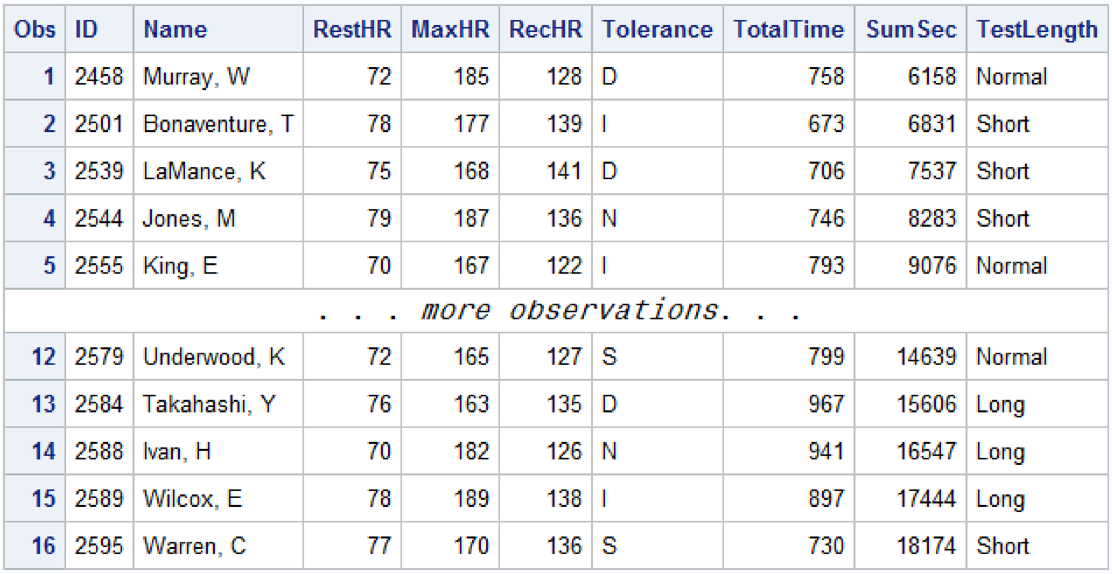
- The DROP= or KEEP= options, in parentheses, follow the names of the data sets that contain the variables to be dropped or kept.
- *variable(s)* identifies the variables to drop or keep.

Example: DROP Data Set Option

Suppose you want to use theTimeMin and TimeSec variables to calculate the total time in the TotalTime variable, but you do not want to keep them in the output data set. You want to keep only the TotalTime variable. When you use the DROP data set option, the TimeMin and TimeSec variables are not written to the output data set:

```
data work.stresstest (drop=timemin timesec);
  set cert.tests;
  if resthr<70 then delete;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
      else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

Output 9.6 *StressTest Data Set with Dropped Variables (partial output)*



Obs	ID	Name	RestHR	MaxHR	RecHR	Tolerance	TotalTime	SumSec	TestLength
1	2458	Murray, W	72	185	128	D	758	6158	Normal
2	2501	Bonaventure, T	78	177	139	I	673	6831	Short
3	2539	LaMance, K	75	168	141	D	706	7537	Short
4	2544	Jones, M	79	187	136	N	746	8283	Short
5	2555	King, E	70	167	122	I	793	9076	Normal
. . . more observations. . .									
12	2579	Underwood, K	72	165	127	S	799	14639	Normal
13	2584	Takahashi, Y	76	163	135	D	967	15606	Long
14	2588	Ivan, H	70	182	126	N	941	16547	Long
15	2589	Wilcox, E	78	189	138	I	897	17444	Long
16	2595	Warren, C	77	170	136	S	730	18174	Short

Another way to exclude variables from a data set is to use the DROP statement or the KEEP statement. Like the DROP= and KEEP= data set options, these statements drop or keep variables. However, the DROP and KEEP statements differ from the DROP= and KEEP= data set options in the following ways:

- You cannot use the DROP and KEEP statements in SAS procedure steps.
- The DROP and KEEP statements apply to all output data sets that are named in the DATA statement. To exclude variables from some data sets but not from others, use the DROP= and KEEP= data set options in the DATA statement.

The KEEP statement is similar to the DROP statement, except that the KEEP statement specifies a list of variables to write to output data sets. Use the KEEP statement instead of the DROP statement if the number of variables to keep is smaller than the number to drop.

Syntax, DROP, and KEEP statements:

DROP *variable(s);*

KEEP *variable(s);*

variable(s) identifies the variables to drop or keep.

Example: Using the DROP Statement

The following example uses the DROP statement to drop unwanted variables.

```
data work.stresstest;
  set cert.tests;
  if tolerance='D';
  drop timemin timesec;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

The Basics of SAS Functions

Definition

SAS *functions* are pre-written routines that perform computations or system manipulations on arguments and return a value. Functions can return either numeric or character results. The value that is returned can be used in an assignment statement or elsewhere in expressions.

Uses of SAS Functions

You can use SAS functions in DATA step programming statements, in WHERE expressions, in macro language statements, in the REPORT procedure, and in Structured Query Language (SQL). They enable you to do the following:

- calculate sample statistics
- create SAS date values
- convert U.S. ZIP codes to state postal codes
- round values
- generate random numbers
- extract a portion of a character value
- convert data from one data type to another

SAS Functions Categories

SAS functions provide programming shortcuts. The following table shows you all of the SAS function categories. This book covers selected functions that convert data, manipulate SAS date values, and modify values of character variables.

Table 14.1 SAS Function Categories

Functions by Category			
Arithmetic	Descriptive Statistics*	Numeric*	State and ZIP code*

Functions by Category			
Array	Distance	Probability	Trigonometric
Bitwise Logical Operations	External Files	Quantile	Truncation*
CAS	External Routines	Random Number	Variable Control
Character*	Financial	SAS File I/O	Variable Information
Character String Matching	Hyperbolic	Search	Web Services
Combinatorial	Macro	Sort	Web Tools
Date and Time	Mathematical*	Special*	

* Denotes the functions that are covered in this Lesson.

SAS Functions Syntax

Arguments and Variable Lists

To use a SAS function, specify the function name followed by the function arguments, which are enclosed in parentheses.

Syntax, SAS function:

function-name(*argument-1<,argument-n>*)**;**

Each of the following are *arguments*:

- variables: mean(*x*,*y*,*z*)
- constants: mean(*456*,*502*,*612*,*498*)
- expressions: mean(37*2,192/5,*mean*(*22*,*34*,*56*))

Note: Even if the function does not require arguments, the function name must still be followed by parentheses (for example, *function-name()*).

When a function contains more than one argument, the arguments are usually separated by commas.

```
function-name(argument-1,argument-2,argument-n);
```

Example: Multiple Arguments

Here is an example of a function that contains multiple arguments. Notice that the arguments are separated by commas.

```
mean(x1,x2,x3)
```

The arguments for this function can also be written as a variable list.

```
mean(of x1-x3)
```

Target Variables

A target variable is the variable to which the result of a function is assigned. For example, in the statement below, the variable AvgScore is the target variable.

```
AvgScore=mean(exam1,exam2,exam3);
```

Unless the length of the target variable has been previously defined, a default length is assigned. The default length depends on the function; the default for character functions can be as long as 200.

TIP Default lengths can cause character variables to use more space than necessary in your data set. So, when using SAS functions, consider the appropriate length for any character target variables. If necessary, add a LENGTH statement to specify a length for the character target variable before the statement that creates the values of that variable.

Manipulating SAS Date Values with Functions

YEAR, QTR, MONTH, and DAY Functions

Overview of YEAR, QTR, MONTH, and DAY Functions

Every SAS date value can be queried for the values of its year, quarter, month, and day. You extract these values by using the functions YEAR, QTR, MONTH, and DAY. They each work the same way.

Syntax, YEAR, QTR, MONTH, and DAY functions:

YEAR(*date***)**

QTR(*date***)**

MONTH(*date***)**

DAY(*date***)**

date is a SAS date value that is specified either as a variable or as a SAS date constant.

The YEAR function returns a four-digit numeric value that represents the year (for example, 2018). The QTR function returns a value of 1, 2, 3, or 4 from a SAS date value to indicate the quarter of the year in which a date value falls. The MONTH function returns a numeric value that ranges from 1 to 12, representing the month of the year. The value 1 represents January, 2 represents February, and so on. The DAY function returns a numeric value from 1 to 31, representing the day of the month.

Table 14.6 Selected Date Functions and Their Uses

Function	Description	Form	Sample Value
YEAR	Extracts the year value from a SAS date value.	YEAR*(date)*	**2018**
QTR	Extracts the quarter value from a SAS date value	QTR*(date)*	**1**
MONTH	Extracts the month value from a SAS date value.	MONTH*(date)*	**12**
DAY	Extracts the day value from a SAS date value	DAY*(date)*	**5**

Example: Finding the Year and Month

Suppose you want to create a subset of the data set Cert.Temp that contains information about all temporary employees who were hired in November 2017. The data set Cert.Temp contains the beginning and ending dates for staff employment, but there are no month or year variables in the data set. To determine the year in which employees were hired, you can apply the YEAR function to the variable that contains the employee start date, StartDate. Here is a way to write the YEAR function:

```
year(startdate)
```

Likewise, to determine the month in which employees were hired, you apply the MONTH function to StartDate.

```
month(startdate)
```

To create the new data set, you include these functions in a subsetting IF statement within a DATA step. The subsetting IF statement specifies the new data set include only the observations where the YEAR function extracts a value of 2017 and the MONTH function extracts a value of 11. The value of 11 stands for November.

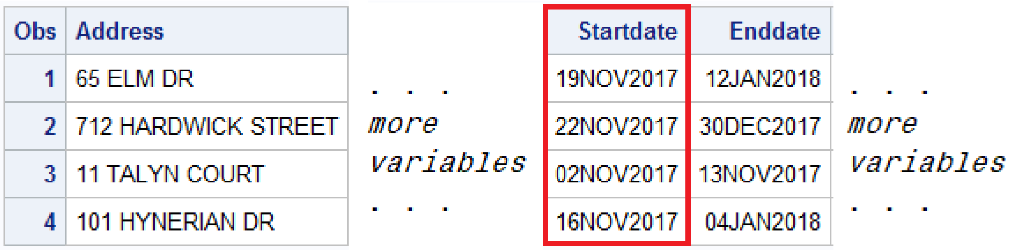
```
data work.nov17;
  set cert.temp;
  if year(startdate)=2017 and month(startdate)=11;
run;
```

When you add a PROC PRINT step to the program, you can view the new data set.

```
proc print data=work.nov17;
  format startdate enddate birthdate date9.;
run;
```

The new data set contains information about only those employees who were hired in November 2017.

__Output 14.3__ PROC PRINT Output of Work.Nov17 (partial output)



Obs	Address	. . . more variables . . .	Startdate	Enddate	. . . more variables . . .
1	65 ELM DR		19NOV2017	12JAN2018	
2	712 HARDWICK STREET		22NOV2017	30DEC2017	
3	11 TALYN COURT		02NOV2017	13NOV2017	
4	101 HYNERIAN DR		16NOV2017	04JAN2018	

Example: Finding the Year

Suppose you want to create a subset of the data set Cert.Temp that contains information about all temporary employees who were hired during a specific year, such as 2016. Cert.Temp contains the dates on which employees began work with the company and their ending dates, but there is no year variable.

To determine the year in which employees were hired, you can apply the YEAR function to the variable that contains the employee start date, StartDate. You write the YEAR function as follows:

```
year(startdate)
```

To create the new data set, you include this function in a subsetting IF statement within a DATA step. This subsetting IF statement specifies that only observations in which the YEAR function extracts a value of **`2016`** are placed in the new data set.

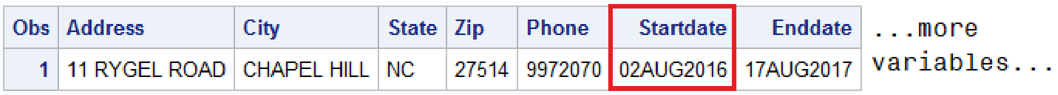
```
data work.temp16;
  set cert.temp;
  if year(startdate)=2016;
run;
```

When you add a PROC PRINT step to the program, you can view the new data set.

```
data work.temp16;
  set cert.temp;
  where year(startdate)=2016;
run;
proc print data=work.temp16;
  format startdate enddate birthdate date9.;
run;
```

The new data set contains information for only those employees who were hired in 2016.

__Output 14.4__ PROC PRINT Output of Work.Temp16 (partial output)



Obs	Address	City	State	Zip	Phone	Startdate	Enddate	...more variables...
1	11 RYGEL ROAD	CHAPEL HILL	NC	27514	9972070	02AUG2016	17AUG2017	

WEEKDAY Function

Overview of the WEEKDAY Function

The WEEKDAY function enables you to extract the day of the week from a SAS date value.

Syntax, WEEKDAY function:

WEEKDAY(*date*)

date is a SAS date value that is specified either as a variable or as a SAS date constant.

The WEEKDAY function returns a numeric value from 1 to 7. The values represent the days of the week.

Table 14.7 Values for the WEEKDAY Function

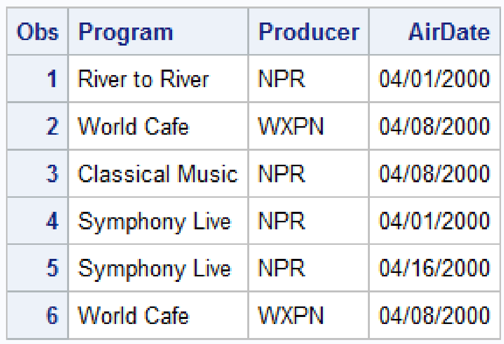
Value	Equals	Day of the Week
1	=	Sunday
2	=	Monday
3	=	Tuesday
4	=	Wednesday
5	=	Thursday
6	=	Friday
7	=	Saturday

Example: WEEKDAY Function

For example, suppose the data set Cert.Sch contains a broadcast schedule. The variable AirDate contains SAS date values. To create a data set that contains only weekend broadcasts, you use the WEEKDAY function in a subsetting IF statement. You include only observations in which the value of AirDate corresponds to a Saturday or Sunday.

```
data work.schwkend;
  set cert.sch;
  if weekday(airdate)in(1,7);
run;
proc print data=work.schwkend;
run;
```

Output 14.5 PROC PRINT Output of Weekday Function



Obs	Program	Producer	AirDate
1	River to River	NPR	04/01/2000
2	World Cafe	WXPN	04/08/2000
3	Classical Music	NPR	04/08/2000
4	Symphony Live	NPR	04/01/2000
5	Symphony Live	NPR	04/16/2000
6	World Cafe	WXPN	04/08/2000

Note: In the example above, the statement **`if weekday(airdate) in (1,7);`** is the same as **`if weekday(airdate)=7 or weekday(airdate)=1;`**

MDY Function

Overview of the MDY Function

The MDY function returns a SAS date value from month, day, and year values.

Syntax, MDY function:

MDY (month, day, year)

- *month* specifies a numeric constant, variable, or expression that represents an integer from 1 through 12.
- *day* specifies a numeric constant, variable, or expression that represents an integer from 1 through 31.
- *year* specifies a numeric constant, variable, or expression with a value of a two-digit or four-digit integer that represents that year.

Example: MDY Function

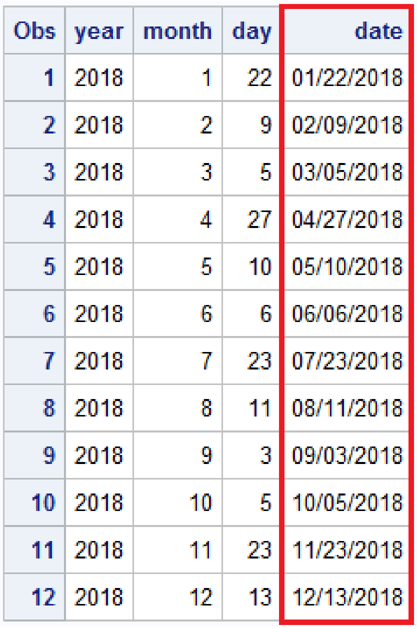
In the data set Cert.Dates, the values for month, day, and year are stored in the numeric variables Month, Day, and Year. It is possible to write the following MDY function to create the SAS date values:

```
mdy(month,day,year)
```

To create a new variable to contain the SAS date values, place this function in an assignment statement.

```
data work.datestemp;
  set cert.dates;
  Date=mdy(month,day,year);
run;
proc print data=work.datestemp;
  format date mmddyy10.;
run;
```

Output 14.6 PROC PRINT Output of Work.Datestemp

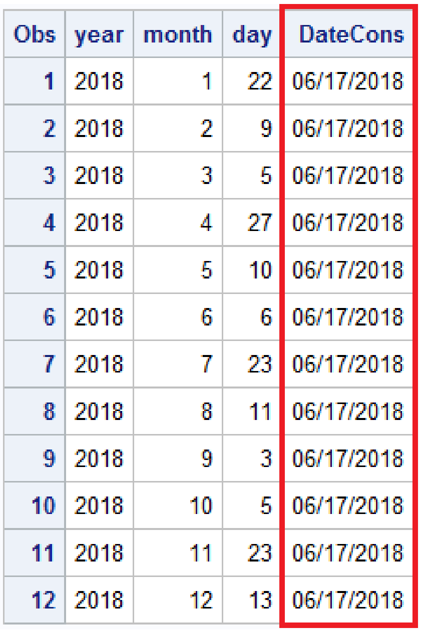


Obs	year	month	day	date
1	2018	1	22	01/22/2018
2	2018	2	9	02/09/2018
3	2018	3	5	03/05/2018
4	2018	4	27	04/27/2018
5	2018	5	10	05/10/2018
6	2018	6	6	06/06/2018
7	2018	7	23	07/23/2018
8	2018	8	11	08/11/2018
9	2018	9	3	09/03/2018
10	2018	10	5	10/05/2018
11	2018	11	23	11/23/2018
12	2018	12	13	12/13/2018

The MDY function can also add the same SAS date to every observation. This might be useful if you want to compare a fixed beginning date with different end dates. Just use numbers instead of data set variables when providing values to the MDY function.

```
data work.datestemp;
  set cert.dates;
  DateCons=mdy(6,17,2018);
run;
proc print data=work.datestemp;
  format DateCons mmddyy10.;
run;
```

Output 14.7 PROC PRINT Output of Work.Datestemp



Obs	year	month	day	DateCons
1	2018	1	22	06/17/2018
2	2018	2	9	06/17/2018
3	2018	3	5	06/17/2018
4	2018	4	27	06/17/2018
5	2018	5	10	06/17/2018
6	2018	6	6	06/17/2018
7	2018	7	23	06/17/2018
8	2018	8	11	06/17/2018
9	2018	9	3	06/17/2018
10	2018	10	5	06/17/2018
11	2018	11	23	06/17/2018
12	2018	12	13	06/17/2018

To display the years clearly, format SAS dates with the DATE9. format. This forces the year to appear with four digits, as shown above in the Date and DateCons variables of the Work.DatesTenp output.

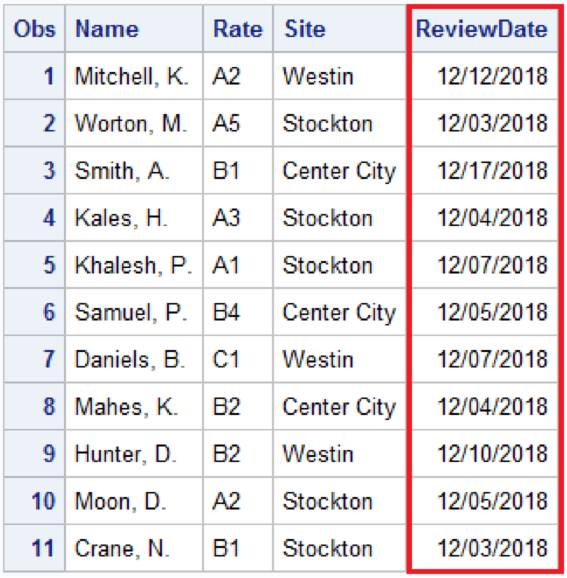
Example: Finding the Date

The data set Cert.Review2018 contains a variable named Day. This variable contains the day of the month for each employee's performance appraisal. The appraisals were all completed in December of 2018.

The following DATA step uses the MDY function to create a new variable named ReviewDate. This variable contains the SAS date value for the date of each performance appraisal.

```
data work.review2018 (drop=Day);
  set cert.review2018;
  ReviewDate=mdy(12,day,2018);
run;
proc print data=work.review2018;
  format ReviewDate mmddyy10.;
run;
```

Output 14.8 *PROC PRINT Output of Work.Review2018*



Obs	Name	Rate	Site	ReviewDate
1	Mitchell, K.	A2	Westin	12/12/2018
2	Worton, M.	A5	Stockton	12/03/2018
3	Smith, A.	B1	Center City	12/17/2018
4	Kales, H.	A3	Stockton	12/04/2018
5	Khalesh, P.	A1	Stockton	12/07/2018
6	Samuel, P.	B4	Center City	12/05/2018
7	Daniels, B.	C1	Westin	12/07/2018
8	Mahes, K.	B2	Center City	12/04/2018
9	Hunter, D.	B2	Westin	12/10/2018
10	Moon, D.	A2	Stockton	12/05/2018
11	Crane, N.	B1	Stockton	12/03/2018

Note: If you specify an invalid date in the MDY function, SAS assigns a missing value to the target variable.

```
data work.review2018 (drop=Day);
  set cert.review2018;
  ReviewDate=mdy(15,day,2018);
run;
proc print data=work.review2018;
  format ReviewDate mmddyy10.;
run;
```

DATE and TODAY Functions

Overview of the DATE Function

The DATE function returns the current date as a numeric SAS date value.

Note: If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the return values for date and time are determined by the time zone.

Syntax, DATE function:

DATE ()

The DATE function does not require any arguments, but it must be followed by parentheses.

The DATE function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

Overview of the TODAY Function

The TODAY function returns the current date as a numeric SAS date value.

Note: If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the return values for date and time are determined by the time zone.

Syntax, TODAY function:

TODAY ()

The TODAY function does not require any arguments, but it must be followed by parentheses.

The TODAY function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

Example: The DATE and TODAY Functions

The DATE and TODAY functions have the same form and can be used interchangeably. To add a new variable, which contains the current date, to the data set Cert.Temp. To create this variable, write an assignment statement such as the following:

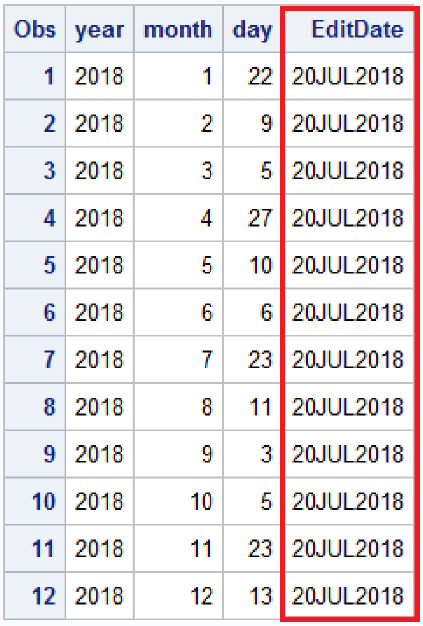
```
EditDate=date();
```

After this statement is added to a DATA step and the step is submitted, the data set that contains EditDate is created. To display these SAS date values in a different form, you can associate a SAS format with the values. For example, the FORMAT statement below associates the DATE9. format with the variable EditDate. The output that is created by this PROC PRINT step appears below.

Note: For this example, the SAS date values shown below were created by submitting this program on July 20, 2018.

```
data work.tempdate;
  set cert.dates;
  EditDate=date();
run;
proc print data=work.tempdate;
  format EditDate date9.;
run;
```

Output 14.9 *PROC PRINT Output of Work.TempDate*



Obs	year	month	day	EditDate
1	2018	1	22	20JUL2018
2	2018	2	9	20JUL2018
3	2018	3	5	20JUL2018
4	2018	4	27	20JUL2018
5	2018	5	10	20JUL2018
6	2018	6	6	20JUL2018
7	2018	7	23	20JUL2018
8	2018	8	11	20JUL2018
9	2018	9	3	20JUL2018
10	2018	10	5	20JUL2018
11	2018	11	23	20JUL2018
12	2018	12	13	20JUL2018

DATDIF and YRDIF Functions

The DATDIF and YRDIF functions calculate the difference in days and years between two SAS dates, respectively. Both functions accept start dates and end dates that are specified as SAS date values. Also, both functions use a basis argument that describes how SAS calculates the date difference.

Syntax, DATDIF, and YRDIF functions:

DATDIF(*start_date,end_date,basis)***)**

YRDIF(*start_date,end_date,basis)***)**

- *start_date* specifies the starting date as a SAS date value.
- *end_date* specifies the ending date as a SAS date value.
- *basis* specifies a character constant or variable that describes how SAS calculates the date difference.

There are two character strings that are valid for basis in the DATDIF function, and four character strings that are valid for basis in the YRDIF function. These character strings and their meanings are listed in the table below.

Table 14.10 *Character Strings in the DATDIF Function*

Character String	Meaning	Valid in DATDIF	Valid in YRDIF
`'30/360'`	specifies a 30-day month and a 360-day year	yes	yes
`'ACT/ACT'`	uses the actual number of days or years between dates	yes	yes
`'ACT/360'`	uses the actual number of days between dates in calculating the number of years (calculated by the number of days divided by 360)	no	yes
`'ACT/365'`	uses the actual number of days between dates in calculating the number of years (calculated by the number of days divided by 365)	no	yes

The best way to understand the different options for the basis argument is to see the different effects that they have on the value that the function returns. The table below lists four YRDIF functions that use the same start date and end date. Each function uses one of the possible values for basis, and each one returns a different value.

Table 14.11 *Examples of the YRDIF Function*

Example Code	Returned Value

`data _null_;`
`  x=yrdif('16feb2016'd,'16jun2018'd,'30/360');`
`  put x;`
`run;`
	`2.3333333333`

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/ACT');`
`  put x;`
`run;`
	`2.3291114604`

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/360');`
`  put x;`
`run;`
	`2.3638888889`

`data _null_;`
`  x=yrdif('16feb2016'd, '16jun2018'd, 'ACT/365');`
`  put x;`
`run;`
	`2.3315068493`

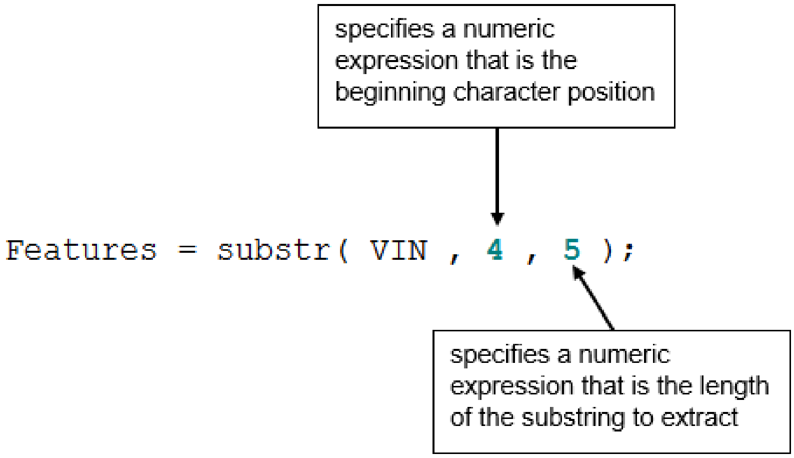
Modifying Character Values with Functions

SUBSTR Function

Overview of the SUBSTR Function

The SUBSTR function extracts a substring from an argument, starting at a specific position in the string.

Figure 14.3 *SUBSTR Function*



specifies a numeric expression that is the beginning character position

```
Features = substr( VIN , 4 , 5 );
```

specifies a numeric expression that is the length of the substring to extract

The SUBSTR function can be used on either the right or left of the equal sign to replace character value constants.

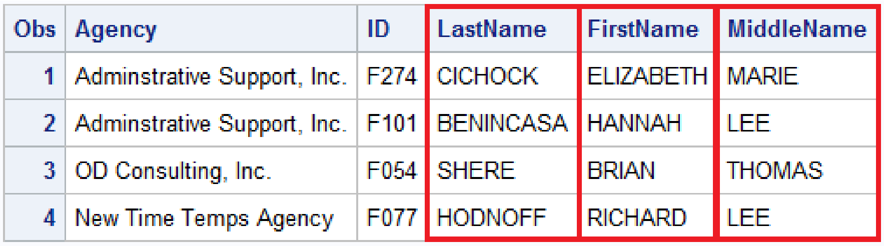
Syntax, SUBSTR function:

SUBSTR(*argument, position* <,n>**)**

- *argument* specifies the character variable or expression to scan.
- *position* is the character position to start from.
- *n* specifies the number of characters to extract. If *n* is omitted, all remaining characters are included in the substring.

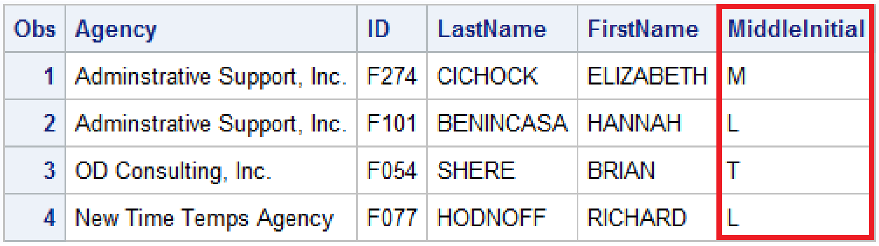
Example: SUBSTR Function

This example begins with the task of extracting a portion of a value. In the data set Cert.AgencyEmp, the names of temporary employees are stored in three name variables: LastName, FirstName, and MiddleName.



Obs	Agency	ID	LastName	FirstName	MiddleName
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	MARIE
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	LEE
3	OD Consulting, Inc.	F054	SHERE	BRIAN	THOMAS
4	New Time Temps Agency	F077	HODNOFF	RICHARD	LEE

However, suppose you want to modify the data set to store only the middle initial instead of the full middle name. To do so, you must extract the first letter of the middle name values and assign these values to the new variable MiddleInitial.



Obs	Agency	ID	LastName	FirstName	MiddleInitial
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	M
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	L
3	OD Consulting, Inc.	F054	SHERE	BRIAN	T
4	New Time Temps Agency	F077	HODNOFF	RICHARD	L

Using the SUBSTR function, you can extract the first letter of the MiddleName value to create the new variable MiddleInitial.

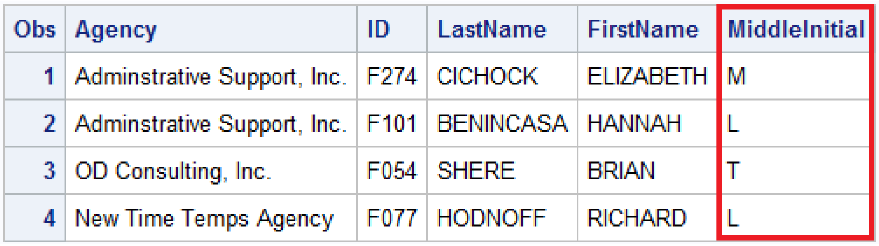
You write the SUBSTR function as the following:

```
substr(middlename,1,1)
```

This function extracts a character string from the value of MiddleName. The string to be extracted begins in position 1 and contains one character. This function is placed in an assignment statement in the DATA step.

```
data work.agencyemp(drop=middlename);
  set cert.agencyemp;
  length MiddleInitial $ 1;
  MiddleInitial=substr(middlename,1,1);
run;
proc print data=work.agencyemp;
run;
```

The new MiddleInitial variable is given the same length as MiddleName. The MiddleName variable is then dropped from the new data set.

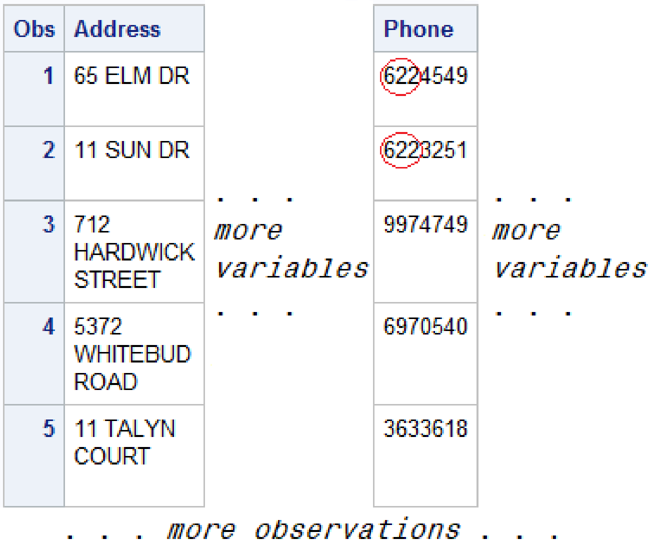


Obs	Agency	ID	LastName	FirstName	MiddleInitial
1	Adminstrative Support, Inc.	F274	CICHOCK	ELIZABETH	M
2	Adminstrative Support, Inc.	F101	BENINCASA	HANNAH	L
3	OD Consulting, Inc.	F054	SHERE	BRIAN	T
4	New Time Temps Agency	F077	HODNOFF	RICHARD	L

You can use the SUBSTR function to extract a substring from any character value if you know the position of the value.

Replacing Text Using SUBSTR

There is a second use for the SUBSTR function. This function can also be used to replace the contents of a character variable. For example, suppose the local phone exchange `622` was replaced by the exchange `433`. You need to update the character variable Phone in Cert.Temp to reflect this change.



Obs	Address	Phone
1	65 ELM DR	6224549
2	11 SUN DR	6223251
3	712 HARDWICK STREET	9974749
4	5372 WHITEBUD ROAD	6970540
5	11 TALYN COURT	3633618

. . . *more variables* . . .

. . . *more variables* . . .

. . . *more observations* . . .

You can use the SUBSTR function to complete this modification. The syntax of the SUBSTR function, when used to replace a variable's values, is identical to the syntax for extracting a substring.

```
SUBSTR(argument,position,n)
```

However, in this case, note the following:

- The first argument specifies the character variable whose values are to be modified.
- The second argument specifies the position at which the replacement is to begin.
- The third argument specifies the number of characters to replace. If *n* is omitted, all remaining characters are replaced.

Positioning the SUBSTR Function

SAS uses the SUBSTR function to extract a substring or to modify a variable's values, depending on the position of the function in the assignment statement.

When the function is on the right side of an assignment statement, the function returns the requested string.

```
MiddleInitial=substr(middlename,1,1);
```

But if you place the SUBSTR function on the left side of an assignment statement, the function is used to modify variable values.

```
substr(region,1,3)='NNW';
```

When the SUBSTR function modifies variable values, the right side of the assignment statement must specify the value to place into the variable. For example, to replace the fourth and fifth characters of a variable named Test with the value **92**, you write the following assignment statement:

```
substr(test,4,2)='92';
```

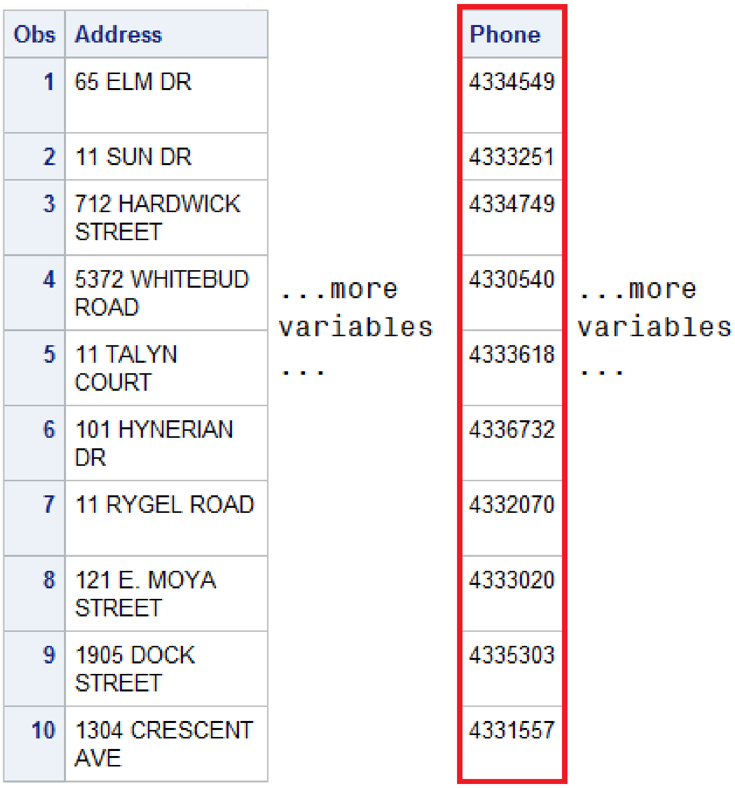
```
Test              Test

S7381K2    →    S7392K2
S7381K7    →    S7392K7
```

It is possible to use the SUBSTR function to replace the **622** exchange in the variable Phone. This assignment statement specifies that the new exchange **433** should be placed in the variable Phone, starting at character position 1 and replacing three characters.

```
data work.temp2;
  set cert.temp;
  substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

However, executing this DATA step places the value **433** into all values of Phone.



Obs	Address		Phone	
1	65 ELM DR		4334549	
2	11 SUN DR		4333251	
3	712 HARDWICK STREET		4334749	
4	5372 WHITEBUD ROAD	...more variables...	4330540	...more variables...
5	11 TALYN COURT		4333618	
6	101 HYNERIAN DR		4336732	
7	11 RYGEL ROAD		4332070	
8	121 E. MOYA STREET		4333020	
9	1905 DOCK STREET		4335303	
10	1304 CRESCENT AVE		4331557	

You need to replace only the values of Phone that contain the **622** exchange. To extract the exchange from Phone, add an assignment statement to the DATA step. Notice that the SUBSTR function is used on the right side of the assignment statement.

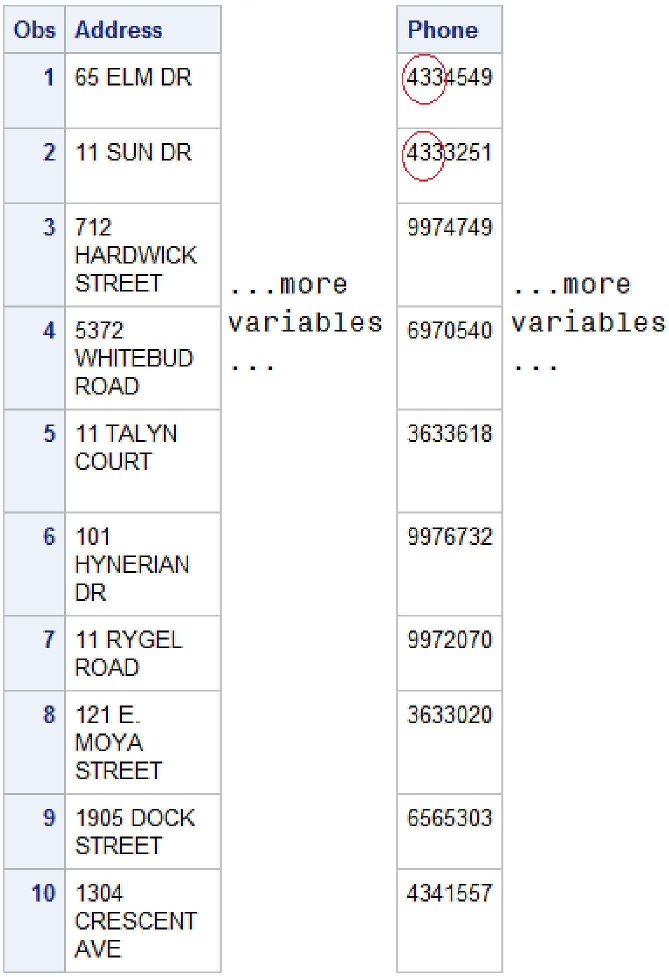
```
data work.temp2(drop=exchange);
  set cert.temp;
  Exchange=substr(phone,1,3);
  substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

Now the DATA step needs an IF-THEN statement to verify the value of the variable Exchange. If the exchange is `622`, the assignment statement executes to replace the value of Phone.

```
data work.temp2(drop=exchange);
  set cert.temp;
  Exchange=substr(phone,1,3);
  if exchange='622' then substr(phone,1,3)='433';
run;
proc print data=work.temp2;
run;
```

After the DATA step is executed, the appropriate values of Phone contain the new exchange.

Figure 14.4 PROC PRINT Output of Work.Temp2 (partial output)



Obs	Address		Phone	
1	65 ELM DR		4334549	
2	11 SUN DR		4333251	
3	712 HARDWICK STREET	...more variables ...	9974749	...more variables ...
4	5372 WHITEBUD ROAD		6970540	
5	11 TALYN COURT		3633618	
6	101 HYNERIAN DR		9976732	
7	11 RYGEL ROAD		9972070	
8	121 E. MOYA STREET		3633020	
9	1905 DOCK STREET		6565303	
10	1304 CRESCENT AVE		4341557	

To summarize, when the SUBSTR function is on the right side of an assignment statement, the function extracts a substring.

```
MiddleInitial=substr(middlename,1,1);
```

When the SUBSTR function is on the left side of an assignment statement, the function replaces the contents of a character variable.

```
 substr(region,1,3)='NNW';
```

CATX Function

Overview of the CATX Function

The CATX function enables you to concatenate character strings, remove leading and trailing blanks, and insert separators. The CATX function returns a value to a variable, or returns a value to a temporary buffer. The results of the CATX function are usually equivalent to those that are produced by a combination of the concatenation operator and the TRIM and LEFT functions.

In the DATA step, if the CATX function returns a value to a variable that has not previously been assigned a length, then the variable is given the length of 200. To save storage space, you can add a LENGTH statement to your DATA step, and specify an appropriate length for your variable. The LENGTH statement is placed before the assignment statement that contains the CATX function so that SAS can specify the length the first time it encounters the variable.

If the variable has not previously been assigned a length, the concatenation operator (||) returns a value to a variable. The variable's given length is the sum of the length of the values that are being concatenated. Otherwise, you can use the LENGTH statement before the assignment statement containing the TRIM function to assign a length.

Recall that you can use the TRIM function with the concatenation operator to create one address variable. The address variable contains the values of the three variables Address, City, and Zip. To remove extra blanks from the new values, use the DATA step shown below:

```
data work.newaddress(drop=address city state zip);
  set cert.temp;
  NewAddress=trim(address)||', '||trim(city)||', '||zip;
run;
```

You can accomplish the same concatenation using only the CATX function.

Syntax, CATX function:

CATX(*separator,string-1* <,...*string-n*>**)**

- *separator* specifies the character string that is used as a separator between concatenated strings
- *string* specifies a SAS character string.

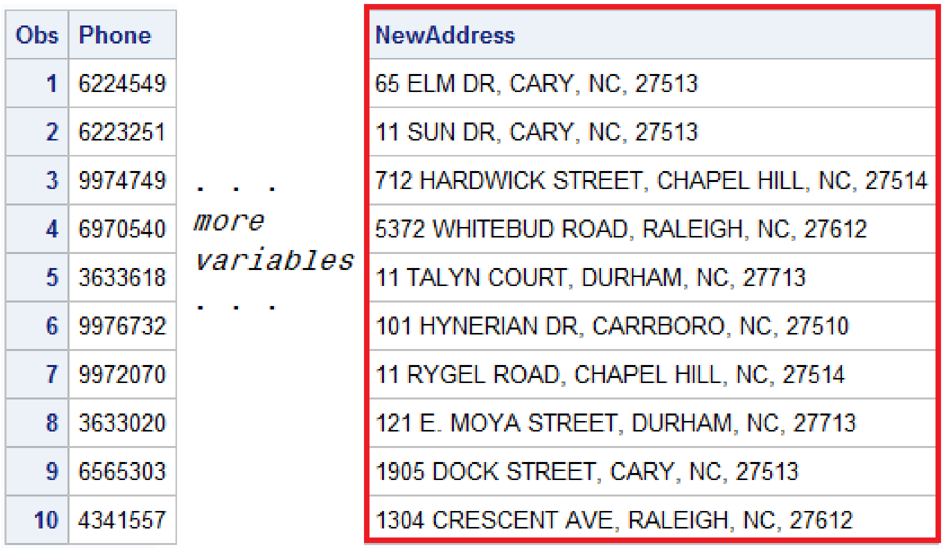
Example: Create New Variable Using CATX Function

You want to create the new variable NewAddress by concatenating the values of the Address, City, and Zip variables from the data set Cert.Temp. You want to strip excess blanks from the old variable's values and separate the variable values with a comma and a space. The DATA step below uses the CATX function to create NewAddress.

```
data work.newaddress(drop=address city state zip);
  set cert.temp;
  NewAddress=catx(', ',address,city,zip);
run;
proc print data=work.newaddress;
run;
```

The revised DATA step creates the values that you would expect for NewAddress.

Output 14.12 *SAS Data Set Work.NewAddress*



Obs	Phone	NewAddress
1	6224549	65 ELM DR, CARY, NC, 27513
2	6223251	11 SUN DR, CARY, NC, 27513
3	9974749	712 HARDWICK STREET, CHAPEL HILL, NC, 27514
4	6970540	5372 WHITEBUD ROAD, RALEIGH, NC, 27612
5	3633618	11 TALYN COURT, DURHAM, NC, 27713
6	9976732	101 HYNERIAN DR, CARRBORO, NC, 27510
7	9972070	11 RYGEL ROAD, CHAPEL HILL, NC, 27514
8	3633020	121 E. MOYA STREET, DURHAM, NC, 27713
9	6565303	1905 DOCK STREET, CARY, NC, 27513
10	4341557	1304 CRESCENT AVE, RALEIGH, NC, 27612

. . . *more variables* . . .

UPCASE Function

The UPCASE function converts all letters in a character expression to uppercase.

Syntax, UPCASE function:

UPCASE(*argument***)**

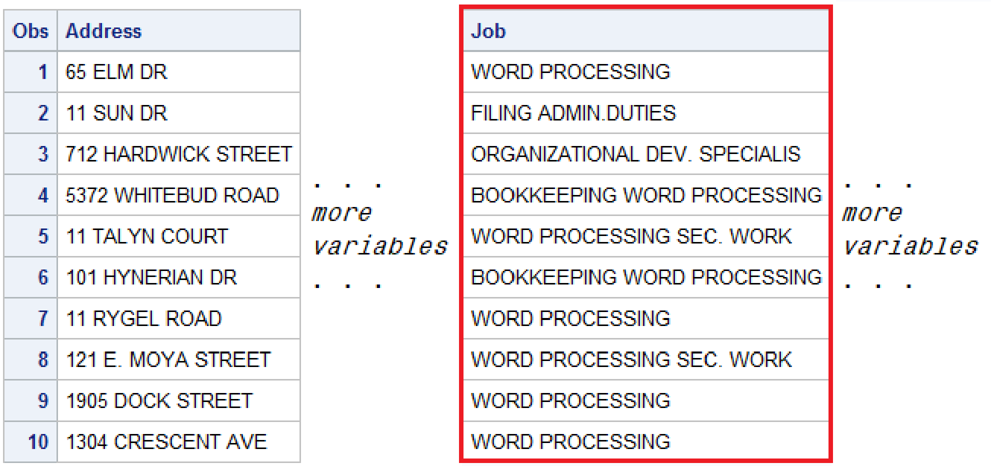
argument can be any SAS character expression, such as a character variable or constant.

In this example, the function is placed in an assignment statement in a DATA step. You can change the values of the variable Job in place.

```
data work.upcasejob;
  set cert.temp;
  Job=upcase(job);
run;
proc print data=work.upcasejob;
run;
```

The new data set contains the converted values of Job.

Output 14.15 *Work.UpcaseJob (partial output)*



Obs	Address	Job
1	65 ELM DR	WORD PROCESSING
2	11 SUN DR	FILING ADMIN.DUTIES
3	712 HARDWICK STREET	ORGANIZATIONAL DEV. SPECIALIS
4	5372 WHITEBUD ROAD	BOOKKEEPING WORD PROCESSING
5	11 TALYN COURT	WORD PROCESSING SEC. WORK
6	101 HYNERIAN DR	BOOKKEEPING WORD PROCESSING
7	11 RYGEL ROAD	WORD PROCESSING
8	121 E. MOYA STREET	WORD PROCESSING SEC. WORK
9	1905 DOCK STREET	WORD PROCESSING
10	1304 CRESCENT AVE	WORD PROCESSING

PROPCASE Function

The PROPCASE function converts all words in an argument to proper case (so that the first letter in each word is capitalized).

Syntax, PROPCASE function:

PROPCASE(*argument<,delimiter(s)>***)**

- *argument* can be any SAS expression, such as a character variable or constant.
- *delimiter(s)* specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

Note: If you specify *delimiter(s)*, then the default delimiters are no longer in effect.

- The PROPCASE function first converts all letters to lowercase letters and then converts the first character of words to uppercase.
- The first character of a word is the first letter of a string or any letter preceded by a default list of delimiters.

  Default delimiter List: blank / — (. tab

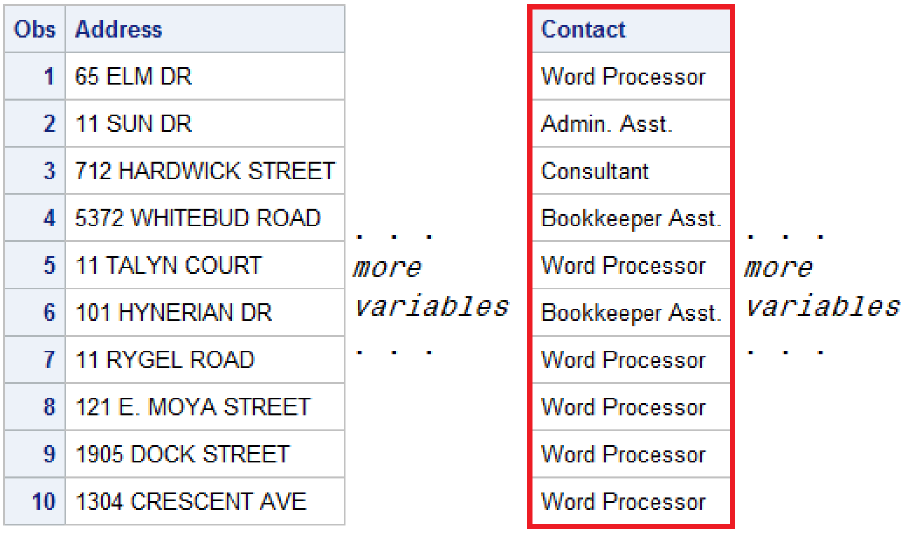
TIP Delimiters can be specified as a second argument, instead of using the default list.

In this example, the function converts the values of the variable named Contact to proper case and uses the default delimiters.

```
data work.propcasecontact;
  set cert.temp;
  Contact=propcase(contact);
run;
proc print data=work.propcasecontact;
run;
```

After the DATA step executes, the new data set is created.

Output 14.17 *Work.PropcaseContact (partial output)*



Obs	Address	Contact
1	65 ELM DR	Word Processor
2	11 SUN DR	Admin. Asst.
3	712 HARDWICK STREET	Consultant
4	5372 WHITEBUD ROAD	Bookkeeper Asst.
5	11 TALYN COURT	Word Processor
6	101 HYNERIAN DR	Bookkeeper Asst.
7	11 RYGEL ROAD	Word Processor
8	121 E. MOYA STREET	Word Processor
9	1905 DOCK STREET	Word Processor
10	1304 CRESCENT AVE	Word Processor

. . . more variables . . .
. . . more variables . . .

Modifying Numeric Values with Functions

SAS provides additional functions to create or modify numeric values. These include arithmetic, financial, and probability functions. This book covers the following selected functions.

CEIL and FLOOR Functions

To return integers that are greater than or equal to the argument, use these functions:

- The CEIL function returns the smallest integer that is greater than or equal to the argument.
- The FLOOR function returns the largest integer that is less than or equal to the argument.

Syntax, CEIL and FLOOR function:

CEIL(*argument***)**

FLOOR(*argument***)**

argument is a numeric variable, constant, or expression.

If the argument is within 1E-12 of an integer, the function returns that integer.

The following SAS statements produce this result:

Table 14.12 CEIL and FLOOR Functions

SAS Statement	Result
CEIL Function Examples	

`data _null_;`
`  var1=2.1;`
`  var2=-2.1;`
`  a=ceil(var1);`
`  b=ceil(var2);`
`  put "a=" a;`
`  put "b=" b;`
`run;`
	`a=3`
	`b=-2`

SAS Statement	Result

data _null_;
  c=ceil(1+1.e-11);
  d=ceil(-1+1e-11);
  e=ceil(1+1.e-13)
  put "c=" c;
  put "d=" d;
  put "e=" e;
run;
	c=2
	d=0
	e=1

data _null_;
  f=ceil(223.456);
  g=ceil(763);
  h=ceil(-223.456);
  put "f=" f;
  put "g=" g;
  put "h=" h;
run;
	f=224
	g=763
	h=-223

FLOOR Function Examples	

data _null_;
  var1=2.1;
  var2=-2.1;
  a=floor(var1);
  b=floor(var2);
  put "a=" a;
  put "b=" b;
run;
	a=2
	b=-3

data _null_;
  c=floor(1+1.e-11);
  d=floor(-1+1e-11);
  e=floor(1+1.e-13)
  put "c=" c;
  put "d=" d;
  put "e=" e;
run;
	c=1
	d=-1
	e=1

data _null_;
  f=floor(223.456);
  g=floor(763);
  h=floor(-223.456);
  put "f=" f;
  put "g=" g;
  put "h=" h;
run;
	f=223
	g=763
	h=-224

INT Function

To return the integer portion of a numeric value, use the INT function. Any decimal portion of the INT function argument is discarded.

Syntax, INT function:

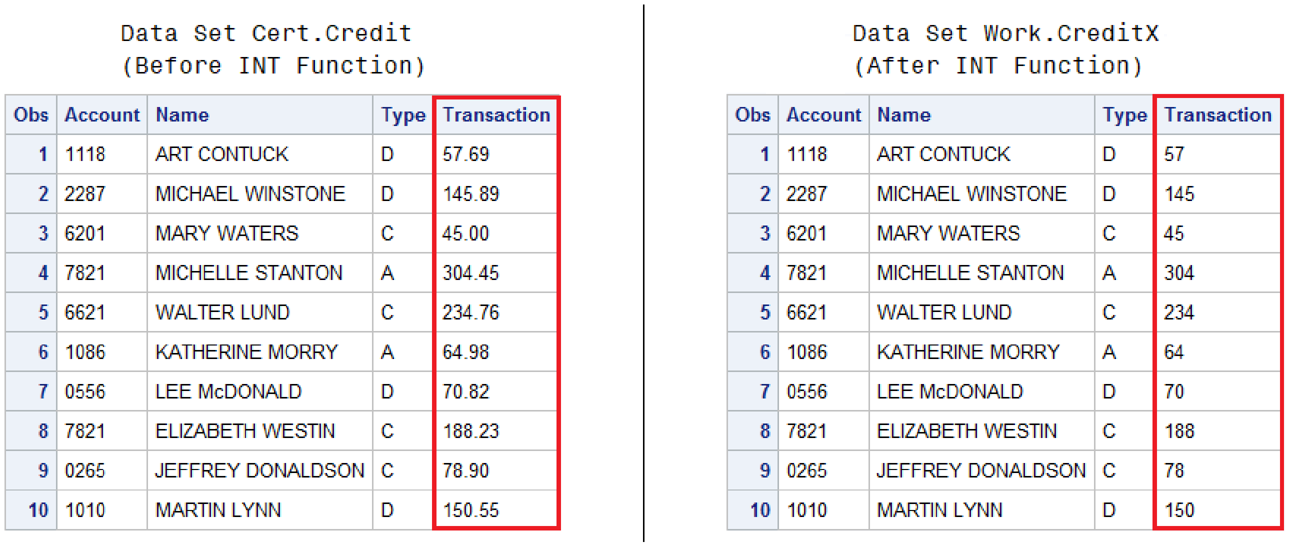
INT(*argument***)**

argument is a numeric variable, constant, or expression.

The two data sets shown below give before-and-after views of values that are truncated by the INT function.

```
data work.creditx;
  set cert.credit;
  Transaction=int(transaction);
run;
proc print data=work.creditx;
run;
```

Output 14.18 *INT Function Comparison*



Data Set Cert.Credit
(Before INT Function)

Obs	Account	Name	Type	Transaction
1	1118	ART CONTUCK	D	57.69
2	2287	MICHAEL WINSTONE	D	145.89
3	6201	MARY WATERS	C	45.00
4	7821	MICHELLE STANTON	A	304.45
5	6621	WALTER LUND	C	234.76
6	1086	KATHERINE MORRY	A	64.98
7	0556	LEE McDONALD	D	70.82
8	7821	ELIZABETH WESTIN	C	188.23
9	0265	JEFFREY DONALDSON	C	78.90
10	1010	MARTIN LYNN	D	150.55

Data Set Work.CreditX
(After INT Function)

Obs	Account	Name	Type	Transaction
1	1118	ART CONTUCK	D	57
2	2287	MICHAEL WINSTONE	D	145
3	6201	MARY WATERS	C	45
4	7821	MICHELLE STANTON	A	304
5	6621	WALTER LUND	C	234
6	1086	KATHERINE MORRY	A	64
7	0556	LEE McDONALD	D	70
8	7821	ELIZABETH WESTIN	C	188
9	0265	JEFFREY DONALDSON	C	78
10	1010	MARTIN LYNN	D	150

ROUND Function

To round values to the nearest specified unit, use the ROUND function.

Syntax, ROUND function:

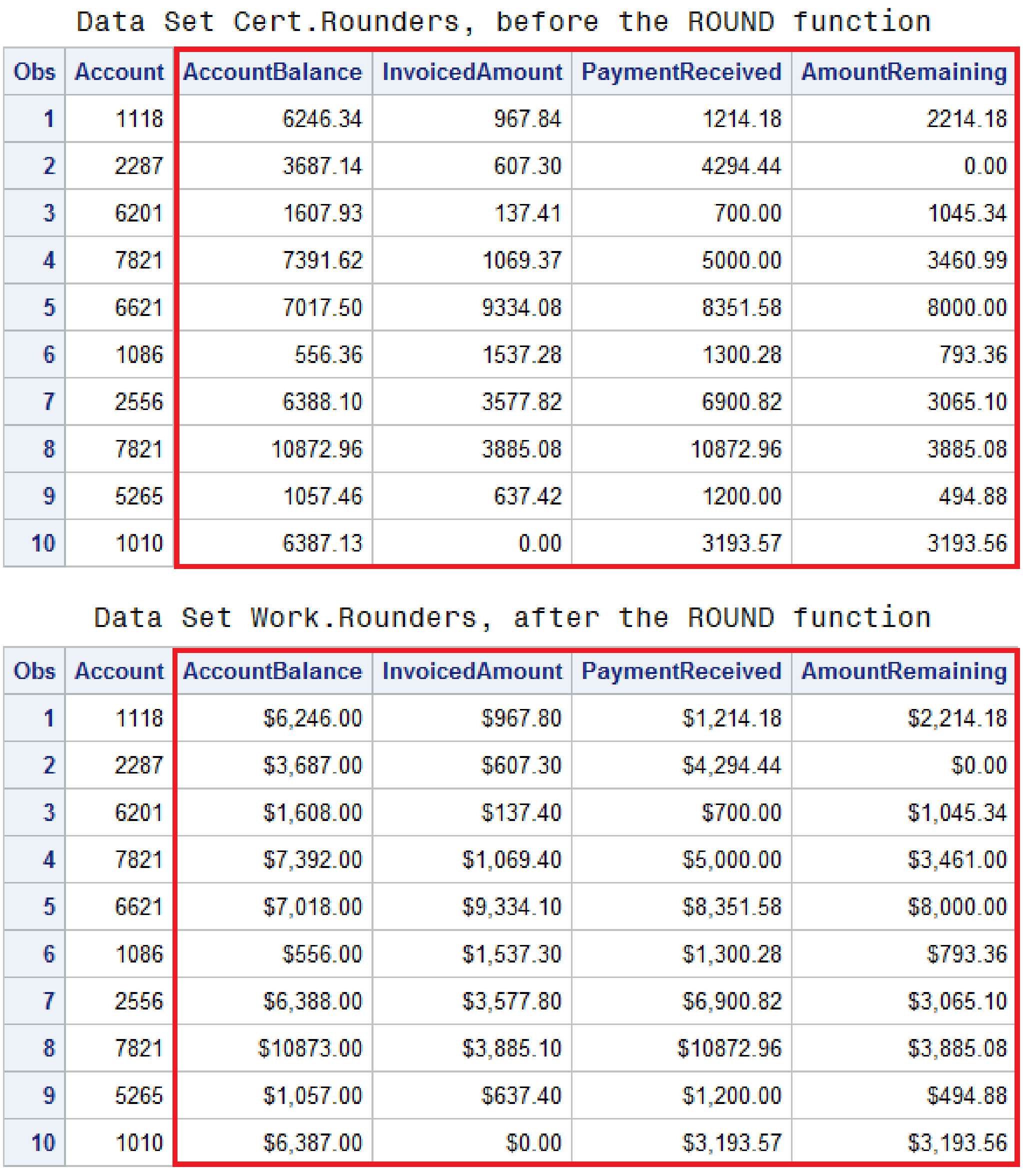
ROUND(*argument,round-off-unit***)**

- *argument* is a numeric variable, constant, or expression.
- *round-off-unit* is numeric and nonnegative.

If a rounding unit is not provided, a default value of 1 is used, and the argument is rounded to the nearest integer. The two data sets shown below give before-and-after views of values that are modified by the ROUND function. The first ROUND function rounds the variable AccountBalance to the nearest integer. The second ROUND function rounds the variable InvoicedAmount to the nearest tenth decimal place. The third ROUND function rounds the variable AmountRemaining to the nearest hundredth decimal place.

```
data work.rounders;
  set cert.rounders;
  AccountBalance=round(AccountBalance, 1);
  InvoicedAmount=round(InvoicedAmount, 0.1);
  AmountRemaining=round(AmountRemaining, 0.02);
  format AccountBalance InvoicedAmount PaymentReceived AmountRemaining dollar9.2;
run;
proc print data=work.rounders;
run;
```

Output 14.19 *Before and After ROUND Function*



Data Set Cert.Rounders, before the ROUND function

Obs	Account	AccountBalance	InvoicedAmount	PaymentReceived	AmountRemaining
1	1118	6246.34	967.84	1214.18	2214.18
2	2287	3687.14	607.30	4294.44	0.00
3	6201	1607.93	137.41	700.00	1045.34
4	7821	7391.62	1069.37	5000.00	3460.99
5	6621	7017.50	9334.08	8351.58	8000.00
6	1086	556.36	1537.28	1300.28	793.36
7	2556	6388.10	3577.82	6900.82	3065.10
8	7821	10872.96	3885.08	10872.96	3885.08
9	5265	1057.46	637.42	1200.00	494.88
10	1010	6387.13	0.00	3193.57	3193.56

Data Set Work.Rounders, after the ROUND function

Obs	Account	AccountBalance	InvoicedAmount	PaymentReceived	AmountRemaining
1	1118	$6,246.00	$967.80	$1,214.18	$2,214.18
2	2287	$3,687.00	$607.30	$4,294.44	$0.00
3	6201	$1,608.00	$137.40	$700.00	$1,045.34
4	7821	$7,392.00	$1,069.40	$5,000.00	$3,461.00
5	6621	$7,018.00	$9,334.10	$8,351.58	$8,000.00
6	1086	$556.00	$1,537.30	$1,300.28	$793.36
7	2556	$6,388.00	$3,577.80	$6,900.82	$3,065.10
8	7821	$10873.00	$3,885.10	$10872.96	$3,885.08
9	5265	$1,057.00	$637.40	$1,200.00	$494.88
10	1010	$6,387.00	$0.00	$3,193.57	$3,193.56