The George Washington University
Department of Statistics

STAT 4197/6197 – Fall 2019

Week 4 – September 20, 2019

Major Topic: Functions, Variable Type Conversions, Arrays, and Do Loops

Detailed Topics:

1. Manipulating Data Using SAS Functions
2. Missing Functions vs. and CALL MISSING Routine
3. Variable Type Conversions (Numeric-to-Character and Character-to Numeric)
4. Using Arrays and Do Loops in SAS

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

Readings:

1. Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012

2. Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015

3. SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition

4. Name that Function: Punny Function Names with Multiple MEANings and Why You Do Not Want to be MISSING Out (By B. Cochran and A.L. Carpenter - SAS Global Forum, 2017)

5. SAS Interview Questions - Functions, etc.

6. INTCK and INTNX: Two essential functions for computing intervals between dates in SAS

SAS Functions

SAS functions can be used in an assignment statement. A *function* is a routine that accepts arguments and returns a value.

```
variable=function-name(argument1, argument2, …);
```

Some functions manipulate character values, compute descriptive statistics, or manipulate SAS date values.

- Arguments are enclosed in parentheses and separated by commas.
- A function can return a numeric or character result.

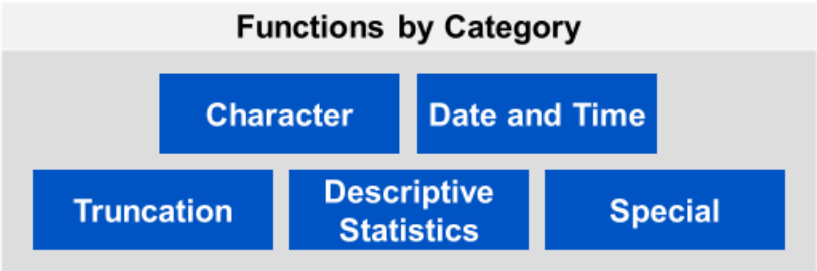
4

SAS Functions by Category

SAS Functions

A SAS *function* performs a computation or system manipulation on arguments and returns a value.

SAS functions can be used in DATA step programming statements and in WHERE expressions.



4

SAS Function by Type of Data Manipulation Performed

SAS Functions

SAS provides a large library of functions for manipulating data during DATA step execution.

A SAS function is often categorized by the type of data manipulation performed:

- Array
- Character
- Date and Time
- Descriptive Statistics
- Financial
- Mathematical
- Probability
- Random Number
- Special
- State and ZIP Code
- Trigonometric

4

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

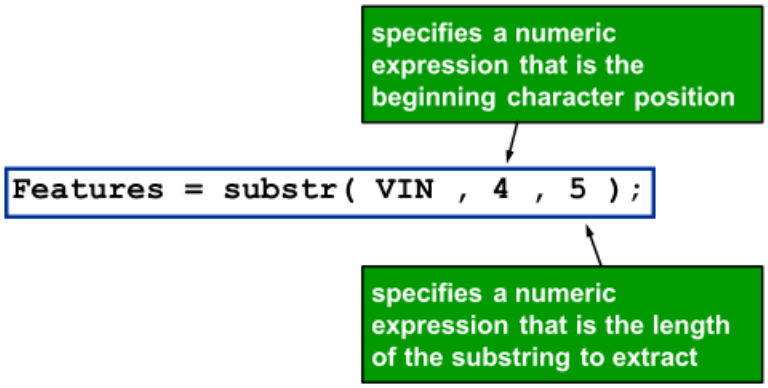
Character Functions

Character		
SUBSTR	CAT	LOWCASE
SCAN	CATS	UPCASE
LEFT	CATT	PROPCASE
RIGHT	CATX	FIND
TRIM	TRANWRD	COMPRESS
STRIP	LENGTH	COMPBL

4

SUBSTR Function

The *SUBSTR function* used to the right of the equal sign extracts a substring from an argument.



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

7

SUBSTR Function

If the SUBSTR function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the length of the first argument.

VIN (17 bytes)	Assignment Statement Using SUBSTR Function	New Variable (17 bytes)
1F1JF27W86J178227	`Make = substr(VIN,2,1);`	F
	`Features = substr(VIN,4,5);`	JF27W
	`SequenceNumber = substr(VIN,12);`	178227

If you omit the length, SAS extracts the remainder of the expression.

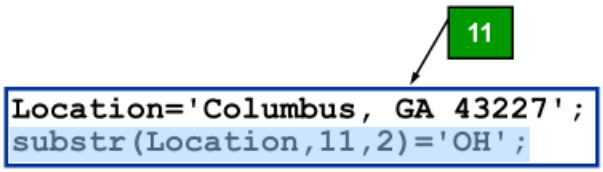
8

The SUBSTR is best used when you know the exact position of the substring to extract from the character value. You specify the variable name, the starting position, and the the number of characters to extract.

SUBSTR Function (Left Side)

This form of the SUBSTR function (left side of assignment statement) replaces characters in a character variable.

Example: Replace two characters starting at position 11.



PDV

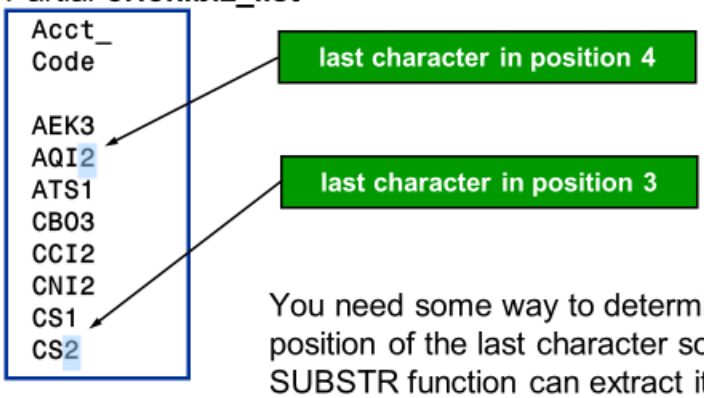
Location
$ 18
Columbus, OH 43227

71

Create the List of Charities: Step 1

The last non-blank character in the **Acct_Code** value occurs in different positions for different observations.

Partial **orion.biz_list**



You need some way to determine the position of the last character so that the SUBSTR function can extract it.

21

LENGTH Function

The LENGTH function returns the length of a non-blank character string, excluding trailing blanks.

General form of the LENGTH function:

NewVar=**LENGTH**(*argument*);

Example:

```
Code='ABCD   ';
Last_NonBlank=length(Code);
```

PDV

Code $ 6	Last_NonBlank N 8
ABCD	4

22

Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Implicit OUTPUT;
Implicit RETURN;

PDV

ID $ 5	Acct_Code $ 6	Name $ 30
AQI	AQI2	AQUAMISSIONS INTERNATIONAL

31

LEFT and RIGHT Functions

- The *LEFT function* left-aligns a character expression.

  LEFT returns an argument with leading blanks moved to the end of the value.

- The *RIGHT function* right-aligns a character expression.

  RIGHT returns an argument with trailing blanks moved to the start of the value.

20

LEFT and RIGHT Functions

What is the value of the new variable?

Var (13 bytes)	Assignment Statement Using the LEFT or RIGHT Function	New Variable (13 bytes)
3 leading blanks ZOOLOGY 3 trailing blanks	`NewVar3 = substr(left(Var),1,3);`	ZOO 10 trailing blanks

23

SCAN Function: Step 1

Extract the second word of **Name**.

NewVar=**SCAN**(*string*,*n*<,*charlist*>);

```
FMName=scan(Name,2,',');
```

PDV

Name $ 16	FMName $ 16
Farr,Sue	Sue

The SCAN function returns the *n*th word of a character value.

48

SCAN Function: Details

When you use the SCAN function:

- A missing value is returned if there are fewer than *n* words in the string.
- If *n* is negative, the SCAN function selects the word in the character string starting from the end of string.
- The length of the created variable is the length of the first argument starting in SAS release 9.4.
- The length of the created variable is 200 bytes in SAS release 9.3 and earlier.
- Delimiters before the first word have no effect.
- Any character or set of characters can serve as delimiters.
- Two or more contiguous delimiters are treated as a single delimiter.

49

1. The variable **Address2** contains values such as *Piscataway, NJ*. Select the statement that extracts and assigns the two-letter state abbreviation to a new variable named **State**.

   a. State=scan(Address2,2);
   b. State=scan(Address2,13,2);
   c. State=substr(Address2,2);
   d. State=substr(Address2,13,2);

The SCAN function is used to extract words from a character when you know the order of the words, when their position varies, and when the words are marked by some delimiter. In this case, you do not need to specify delimiters, because the blank and the comma are default delimiters.

158

Creating Mailing List Data

Using the SCAN function gives an easy way to separate the names for the mailing list.

```
data labels;
   set orion.contacts;
   length FMName LName $ 15;
   FMName=scan(Name,2,',');
   LName=scan(Name,1,',');
run;
```

Partial **labels** data set

ID	Name	Title	FMName	LName
AQI	Farr,Sue	Ms.	Sue	Farr
CCI	Cox,Kay B.	Dr.	Kay B.	Cox
CNI	Mason,Ron	Mr.	Ron	Mason
CS	Ruth,G. H.	Ms.	G. H.	Ruth

54 p205d06

Concatenation Operator

The *concatenation operator* (|| - two vertical bars, ¦¦ - two broken vertical bars, or !! - two exclamation points) concatenates character values.

```
FullName = First || Middle || Last;
```

- The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation, unless you use a LENGTH statement to specify a different length for the new variable.
- The concatenation operator does not trim leading or trailing blanks. If variables are padded with trailing blanks, use the TRIM function to trim trailing blanks from values before concatenating them.

24

TRIM Function

The *TRIM function* removes trailing blanks from character expressions and returns one blank if the expression is missing.

```
FullName = trim(First) || trim(Middle) || Last;
```

- The TRIM function is useful for concatenating because concatenation does not remove trailing blanks.
- If the TRIM function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the length of the argument.

26

TRIM Function

```
data name;
   length Name $ 20 First Middle Last $ 10;
   Name = 'Jones, Mary Ann, Sue';
   First = left(scan(Name,2,','));
   Middle = left(scan(Name,3,','));
   Last = scan(name,1,',');
   FullName = trim(First)||trim(Middle)||Last;
run;
```

First (10 bytes)	Middle (10 bytes)	Last (10 bytes)
Mary Ann	Sue	Jones

FullName (30 bytes)
Mary AnnSueJones

14 blanks

27

TRIM Function

The following program is submitted:

```
data name;
   length Name $ 20 First Middle Last $ 10;
   Name = 'Jones, Mary Ann, Sue';
   First = left(scan(Name,2,','));
   Middle = left(scan(Name,3,','));
   Last = scan(name,1,',');
   Name1=trim(Middle);
   Name2=trim(First)||' '||trim(Middle)||' '||Last;
run;
```

What is the byte size of **Name1** and **Name2**?

A. Name1=3 and Name2=30

B. Name1=3 and Name2=32

C. Name1=10 and Name2=30

D. Name1=10 and Name2=32

29

STRIP Function

The *STRIP function* removes leading and trailing blanks from character expressions.

```
data name;
   length Name $ 20 Last First Middle $ 10;
   Name = 'Jones, Mary Ann, Sue';
   Last = scan(name,1,',');
   First = scan(Name,2,',');
   Middle = scan(Name,3,',');
   Name2=strip(First)||' '||strip(Middle)||
         ' '||Last;
run;
```

- This example does not use the LEFT function when creating the **First** and **Middle** variables.

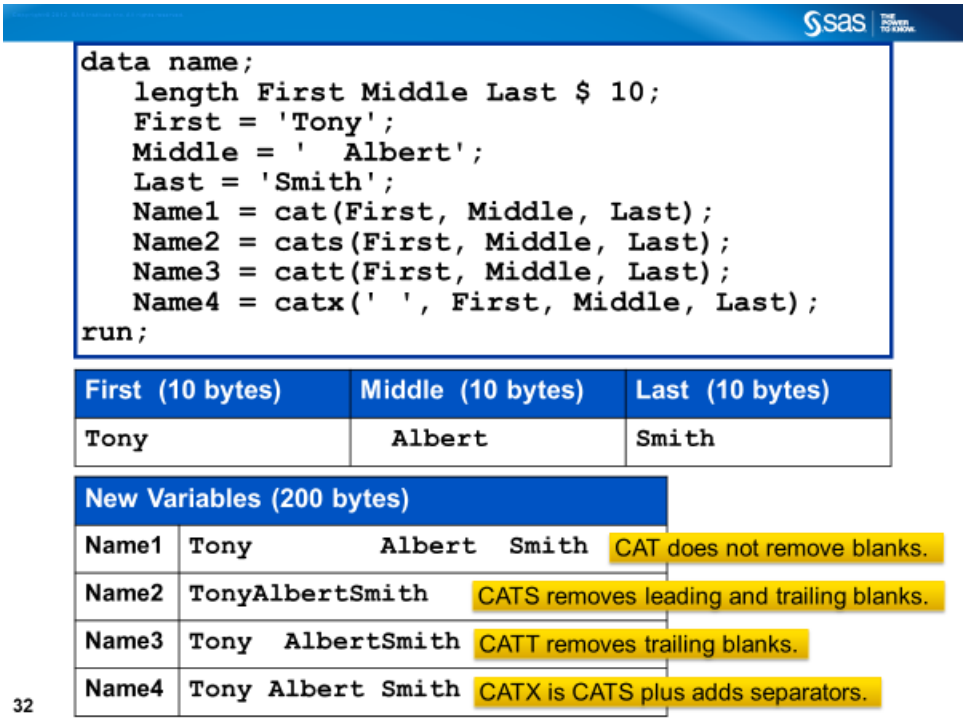
30

CAT Functions

The following functions concatenate character strings:

CAT	Does not remove leading or trailing blanks.
	`FullName1 = cat(First, Middle, Last);`
CATS	Removes leading and trailing blanks.
	`FullName2 = cats(First, Middle, Last);`
CATT	Removes trailing blanks.
	`FullName3 = catt(First, Middle, Last);`
CATX	Removes leading and trailing blanks and inserts separators.
	`FullName4 = catx(' ', First, Middle, Last);`

31

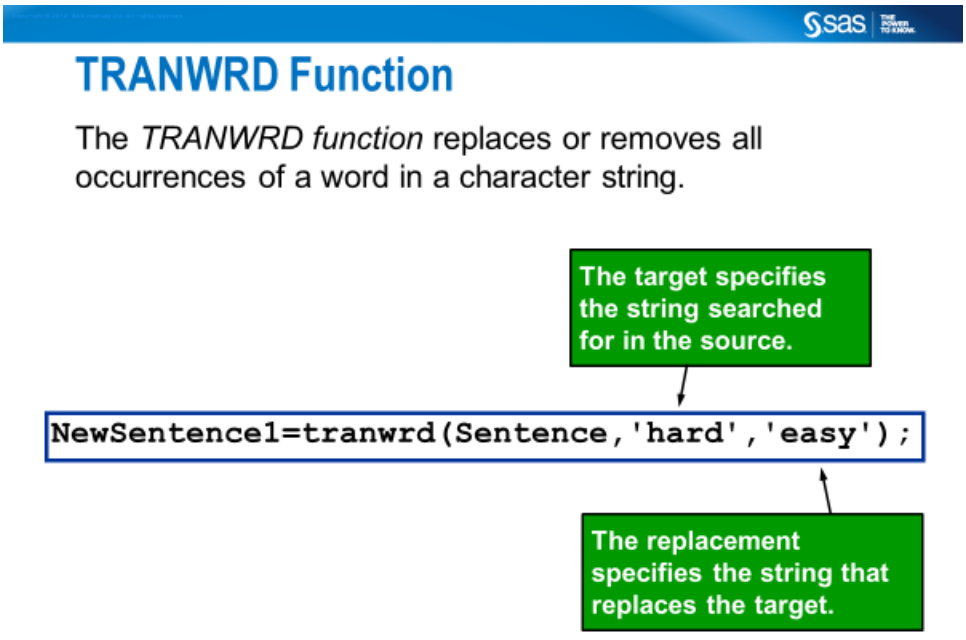


```
data name;
   length First Middle Last $ 10;
   First = 'Tony';
   Middle = '  Albert';
   Last = 'Smith';
   Name1 = cat(First, Middle, Last);
   Name2 = cats(First, Middle, Last);
   Name3 = catt(First, Middle, Last);
   Name4 = catx(' ', First, Middle, Last);
run;
```

First (10 bytes)	Middle (10 bytes)	Last (10 bytes)
Tony	Albert	Smith

New Variables (200 bytes)		
Name1	Tony Albert Smith	CAT does not remove blanks.
Name2	TonyAlbertSmith	CATS removes leading and trailing blanks.
Name3	Tony AlbertSmith	CATT removes trailing blanks.
Name4	Tony Albert Smith	CATX is CATS plus adds separators.

32



TRANWRD Function

The *TRANWRD function* replaces or removes all occurrences of a word in a character string.

The target specifies the string searched for in the source.

```
NewSentence1=tranwrd(Sentence,'hard','easy');
```

The replacement specifies the string that replaces the target.

35

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

TRANWRD Function

If the TRANWRD function returns a value to a variable that was not yet assigned a length, by default, the variable is assigned a length of 200.

Sentence (31 bytes)
`Functions are very hard to use.`

Assignment Statement Using the TRANWRD Function	New Variable (200 bytes)
`NewSentence1 = tranwrd(Sentence,'hard','easy');`	`Functions are very easy to use.`
`NewSentence2 = tranwrd(Sentence,'hard','difficult');`	`Functions are very difficult to use.`

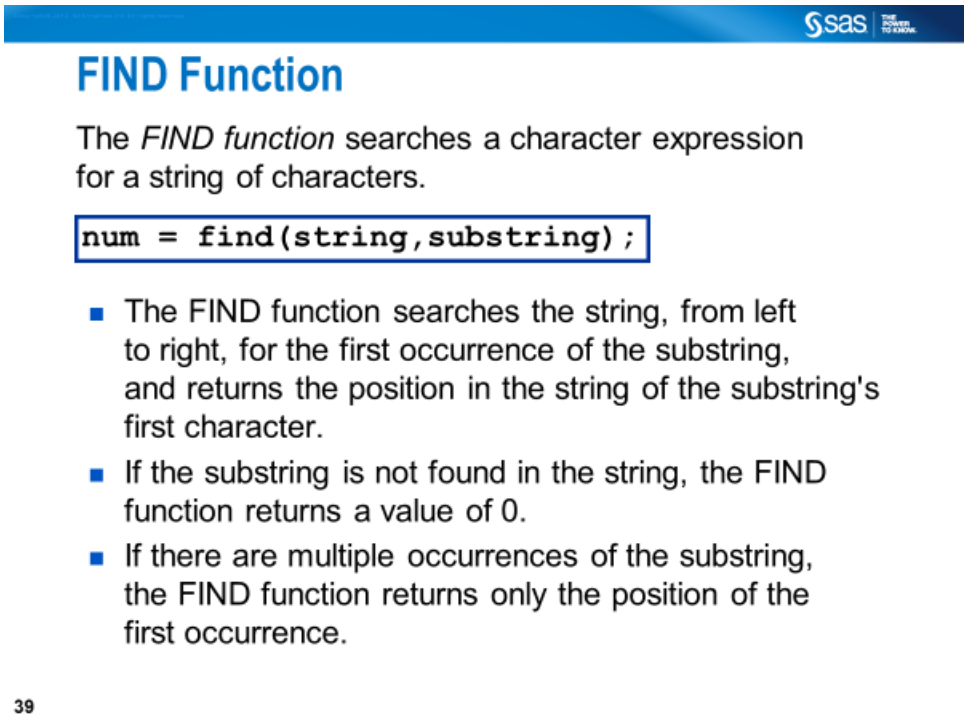
36

Partial SAS Log)

```
date1=12/31/2010
date1_translate=12-31-2010
txt=Data from surveys
txt_tranwrd=Data from records
```

"TRANSLATE handles character replacement for single-byte character sets only.

The TRANWRD function differs from TRANSLATE in that it scans for words (or patterns of characters) and replaces those words with a second word (or pattern of characters)." SAS Documentation.



FIND Function

A *modifier* is a character constant, variable, or expression that specifies one or more modifiers. The following modifiers can be in uppercase or lowercase:

- i - ignores character case during the search.
- t - trims trailing blanks from string and substring.

46

continued...

FIND Function

Ignore case and trim trailing blanks.

```
num5 = find(string,'WOOD ','it',15);
```

Start at position 15 and search to the right.

string character 37 bytes	num5 numeric 8 bytes
How much WOOD would a woodchuck chuck	23

15 → 23

48

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

Case Functions

The following program is submitted:

```
data example;
   Var='r&d, u.s. division (not puerto rico)';
   NewVar=propcase(Var);
run;
```

The following is the desired value of **NewVar**:

```
R&D, U.S. Division (Not Puerto Rico)
```

Yes or No? Does the program create the desired value of **NewVar**?

```
R&d, U.S. Division (Not Puerto Rico)
```

55

COMPBL and COMPRESS Functions

- The *COMPBL function* removes multiple blanks in a character string by translating each occurrence of two or more consecutive blanks into a single blank.
- The *COMPRESS function* returns a character string with specified characters removed from the string.

var (13 bytes)	Assignment Statement	New Variable (13 bytes)
ABC - DEF GH	Var1=compbl(Var);	ABC - DEF GH
	Var2=compress(Var);	ABC-DEFGH
	Var3=compress(Var,'-');	ABC DEF GH
	Var4=compress(Var,'- ');	ABCDEFGH

56

COMPRESS Function: Example

The COMPRESS function removes the characters listed in the *chars* argument from the *source*.

```
ID='20 01-005 024';
New_ID1=compress(ID);
New_ID2=compress(ID,'-');
New_ID3=compress(ID,' -');
```

PDV

ID $ 13	New_ID1 $ 13
20 01-005 024	2001-005024

New_ID2 $ 13	New_ID3 $ 13
20 01005 024	2001005024

78

Other Functions That Remove Blanks

Function	Purpose
TRIM(*string*)	Removes trailing blanks from a character string.
STRIP(*string*)	Removes all leading and trailing blanks from a character string.
COMPBL(*string*)	Removes multiple blanks from a character string by translating each occurrence of two or more consecutive blanks into a single blank.

79

- COMPRESS function removes unwanted characters from a string variable.
- COMPBL function replaces multiple blanks with single blanks.

Date Functions

- The *WEEKDAY function* returns the day of the week (1=Sunday, … 7=Saturday) from a SAS date value.
- The *DAY function* returns the day of the month (1-31) from a SAS date value.
- The *MONTH function* returns the month (1-12) from a SAS date value.
- The *QTR function* returns the quarter of the year (1-4) from a SAS date value.
- The *YEAR function* returns the four-digit year from a SAS date value.

62

Date Functions: Creating SAS Dates

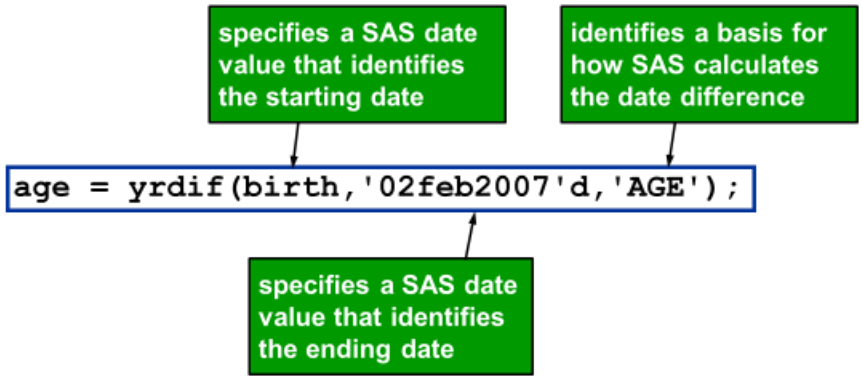
Syntax	Description
TODAY() DATE()	Returns the current date as a SAS date value.
MDY(*month*,*day*,*year*)	Returns a SAS date value from numeric month, day, and year values.

Examples
CurrentDate=today();
y2k=mdy(01,1,2000);
NewYear=mdy(Mon,Day,2013);

8

YRDIF Function

The *YRDIF function* returns the difference in years between two dates.



If the value of basis is AGE, then YRDIF computes the age. The age computation takes into account leap years.

67

YRDIF Function

```
data age;
   set birthday;
   age = yrdif(birth,'02feb2007'd,'AGE');
run;
```

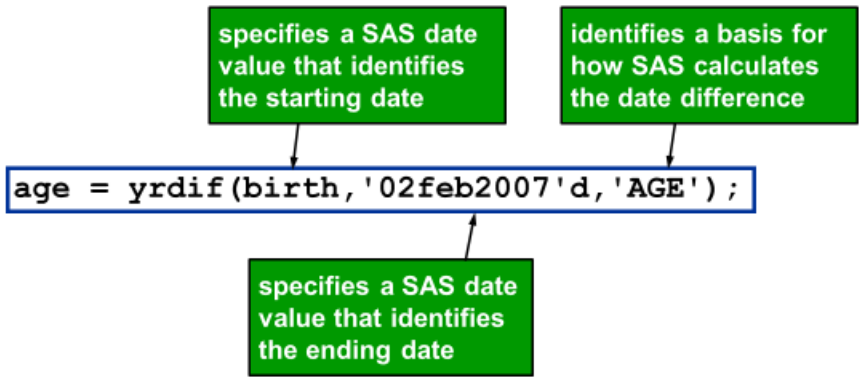
	birth	age
1	3985	36.178082192
2	3673	37.032876712

3985 is November 29, 1970.
3673 is January 21, 1970.

68

YRDIF Function

The *YRDIF function* returns the difference in years between two dates.



If the value of basis is AGE, then YRDIF computes the age. The age computation takes into account leap years.

67

YRDIF Function

Given the following assignment statement:

```
age=yrdif('05MAY1999'd,'10NOV1999'd,'AGE');
```

What is an approximate value of **age**? 0.5178

70

The INTCK Function

The INTCK function returns the integer count of the number of interval boundaries between two dates, two times, or two datetime values.

The INTNX Function

Use this function to advance a date, say month(s), into the future using the following alignment position

- Beginning (b) interval start
- Middle (m) interval center
- End (e) interval end
- Same (s) same relative position as the initial interval

Truncation Functions

Truncation
CEIL
FLOOR
INT
ROUND

71

Exercise 3 Solution

num	ceil(num)	floor(num)	int(num)	round(num)
2.75	3	2	2	3
-2.75	-2	-3	-2	-3
23.1234	24	23	23	23
-23.1234	-23	-24	-23	-23

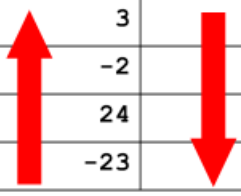
73

CEIL and FLOOR 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.

num	ceil(num)	floor(num)
2.75	3	2
-2.75	-2	-3
23.1234	24	23
-23.1234	-23	-24



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

74

INT Function

The *INT function* returns the integer value.

num	int(num)
2.75	2
-2.75	-2
23.1234	23
-23.1234	-23

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

77

INT Function

Yes or No: Are both of the following statements ***true***?

- The INT function has the same result as the FLOOR function if the value of the argument is positive.
- The INT function has the same result as the CEIL function if the value of the argument is negative.

num	ceil(num)	floor(num)	int(num)
2.75	3	2	2
-2.75	-2	-3	-2
23.1234	24	23	23
-23.1234	-23	-24	-23

79

ROUND Function

The *ROUND function* rounds the first argument to the nearest integer when the second argument is omitted.

num	round(num)
2.75	3
-2.75	-3
23.1234	23
-23.1234	-23

80

ROUND Function

The ROUND function rounds the first argument to the nearest multiple of the second argument.

```
data rounding;
   d1 = round(1234.56789 , 100);
   d2 = round(1234.56789 , 10);
   d3 = round(1234.56789 , 1);
   d4 = round(1234.56789 , .1);
   d5 = round(1234.56789 , .01);
   d6 = round(1234.56789 , .001);
run;
```

d1	d2	d3	d4	d5	d6
1200	1230	1235	1234.6	1234.57	1234.568

81

Descriptive Statistics Functions

MAX	Returns the largest value.
MEAN	Returns the arithmetic mean (average).
MIN	Returns the smallest value.
SUM	Returns the sum of the nonmissing arguments.
N	Returns the number of nonmissing numeric values.
NMISS	Returns the number of missing numeric values.
CMISS	Returns the number of missing numeric and character values.

84

Descriptive Statistics Functions

```
data math;
   var1=2;
   var2=6;
   var3=.;
   var4=4;
   maximum = max(var1,var2,var3,var4);
   average = mean(var1,var2,var3,var4);
   minimum = min(var1,var2,var3,var4);
   total = sum(var1,var2,var3,var4);
run;
```

maximum	average	minimum	total
6	4	2	12

1

Descriptive Statistics Functions

The argument list can consist of a variable list, which is preceded by OF.

```
data math;
   var1=2;
   var2=6;
   var3=.;
   var4=4;
   maximum = max(of var1-var4);
   average = mean(of var1-var4);
   minimum = min(of var1-var4);
   total = sum(of var1-var4);
run;
```

maximum	average	minimum	total
6	4	2	12

86

SAS Variable Lists (Review)

An alternative method to entering variable names separately is to use a *SAS variable list*.

```
data contrib;
   set orion.employee_donations;
   Total=sum(of Qtr1-Qtr4);
   if Total ge 50;
run;
```

- The keyword OF must precede the variable list.

8 p205d01

SAS Variable Lists: Examples

Numbered Range List

PDV

Qtr1	Qtr2	Var1	Qtr3	Qtr4

```
Total=sum(of Qtr1-Qtr4)
```

Var1 not included

Name Range List

PDV

Qtr1	Second	Q3	Fourth	Var2

```
Total=sum(of Qtr1--Fourth)
```

Var2 not included

9

SAS Variable Lists: Examples

Name Prefix List

PDV

TotJan	Qtr2	TotFeb	TotMar

```
Total=sum(of Tot:)
```

Qtr2 not included

Special Name Lists

PDV

Qtr1 N	Name $	Q3 N	Fourth N

```
Total=sum(of _Numeric_)
```

Name not included

10

Using SAS Functions (Review)

You can use functions in DATA step statements anywhere that an expression can appear.

```
data contrib;
   set orion.employee_donations;
   Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   if Total ge 50;
run;

proc print data=contrib noobs;
   title 'Contributions $50 and Over';
   var Employee_ID Qtr1 Qtr2 Qtr3 Qtr4
   Total;
run;
```

function-name(argument-1,argument-2,…,argument-n)

6

p205d01

Missing Values

- Numeric missing values are represented as a single period (.).
- Numeric special missing values can be represented by a single period plus a letter or an underscore (.A - .Z or ._).
- Character missing values are represented as a blank ( ).

Use the following functions to count missing values for multiple variables in the argument list

- NMISS() for numeric variables
- CMISS() for numeric and character variables

Use the following functions to count nonmissing values for multiple numeric variables in the argument list

- N() for numeric variables

The CALL MISSING routine assigns an ordinary numeric missing value (.) to each numeric variable in the argument list. The same routine does also assign a character missing value (a blank) to each character variable in the argument list. [SAS® Documentation]

Use the MISSING () function to identify whether the value of a single variable is missing

- Missing() returns the value 1 or 0

Automatic Character-to-Numeric Conversion

Automatic character-to-numeric conversion happens when a character value is used in a numeric context.

For example:

- assignment to a numeric variable

```
num=char;
```

- an arithmetic operation

```
num2=num1+char;
```

- logical comparison with a numeric value

```
if num>char;
```

- a function that takes numeric arguments

```
num2=mean(num1,char);
```

91

Automatic Character-to-Numeric Conversion

Automatic character-to-numeric conversion

- uses the W. informat
- produces a numeric missing if the character value does not conform to the W. informat
- writes a message to the SAS log stating that the conversion occurred.

```
819  data numeric;
820    num1=5;
821    char='6';
822    num2=num1+char;
823  run;

NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      822:13
NOTE: The data set WORK.NUMERIC has 1 observations and 3 variables.
```

92

Automatic Numeric-to-Character Conversion

Automatic numeric-to-character conversion happens when a numeric value is used in a character context.

For example:

- assignment to a character variable

```
char=num;
```

- a concatenation operation

```
char2=char1||num;
```

- a function that takes character arguments

```
char=substr(num,3,1);
```

93

Automatic Numeric-to-Character Conversion

Automatic numeric-to-character conversion

- uses the BEST12. format
- right-aligns the resulting character value
- writes a message to the SAS log stating that the conversion occurred.

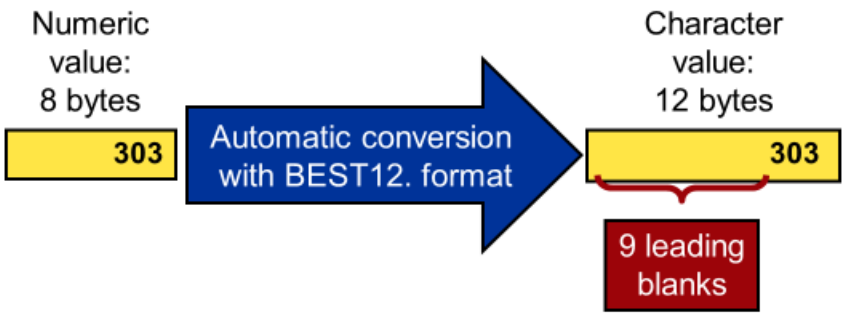
```
828  data character;
829    num=1234567;
830    char=substr(num,3,1);
831  run;

NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
      830:15
NOTE: The data set WORK.CHARACTER has 1 observations and 2 variables.
```

94

Automatic Numeric-to-Character Conversion

The automatic conversion
- uses the BEST12. format
- right-aligns the resulting character value.



144

Automatic Numeric-to-Character Conversion

```
data hrdata;
   keep Phone Code Mobile;
   set orion.convert;
   Phone='(' !! Code !! ') ' !! Mobile;
run;
```

Partial PDV

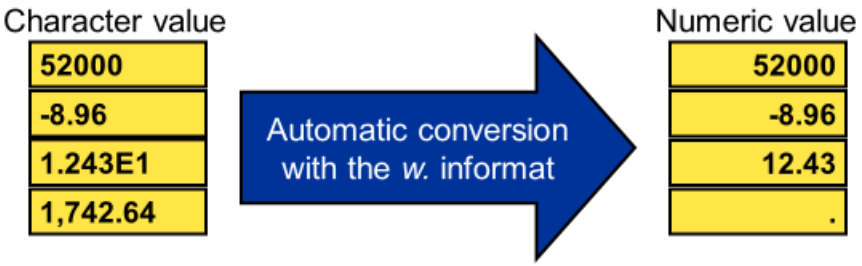
Phone
$ 23
(303) 393-0956

9 leading blanks

To fix this, use the PUT function to explicitly control the numeric-to-character conversion.

145

Automatic Character-to-Numeric Conversion



- The values in **GrossPay** contain commas, which cannot be converted by the *w.* informat, so **GrossPay** is assigned a missing value.
- To explicitly convert the values in **GrossPay**, use the INPUT function.

117

Exercise 4 Solution

Value of Character Variable
162400
$162,400
49275.937
+24
-73.5
01234
52E3
01/01/1960

Automatic character-to-numeric conversion using *W.* informat

Value of Numeric Variable (8 bytes)
162400
.
49275.937
24
-73.5
1234
52000
.

96

Exercise 4 Solution

Value of Numeric Variable (8 bytes)
162400
49275.937
-73.5
52E3
0

Automatic numeric-to-character conversion using BEST12. format

Value of Character Variable (12 bytes)
162400
49275.937
-73.5
52000
0

leading blanks

97

Explicit Conversion Using SAS Functions

Explicit conversion using a SAS function

- produces desirable results
- does ***not*** write a message to the SAS log stating that the conversion occurred.

Special	
INPUT	PUT
character-to-numeric	numeric-to-character

98

Converting Character and Numeric Data

Data can be converted with the following two methods:

- automatic conversion
- explicit conversion with a SAS function

Special	
INPUT	PUT
character-to-numeric	numeric-to-character

90

Explicit Conversion Using SAS Functions

The INPUT function converts a character value to a numeric value.

- The second argument is a numeric informat.
- If the INPUT function returns a value to a variable that was not yet assigned a length, by default, the variable length is 8 bytes.

The PUT function converts a numeric value to a character value.

- The second argument is a numeric format.
- If the PUT function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the width of the format.

99

INPUT Function

Value of Character Variable		Value of Numeric Variable (8 bytes)
162400	input('162400',6.)	162400
$162,400	input('$162,400',comma8.)	162400
49275.937	input('49275.937',9.)	49275.937
+24	input('+24',3.)	24
-73.5	input('-73.5',5.)	-73.5
01234	input('01234',5.)	1234
52E3	input('52E3',4.)	52000
01/01/1960	input('01/01/1960',mmddyy10.)	0

100

PUT Function

Value of Numeric Variable (8 bytes)		Value of Character Variable
162400	put(162400,dollar8.);	$162,400
49275.937	put(49275.937,comma10.3);	49,275.937
-73.5	put(-73.5,5.1);	-73.5
52E3	put(52E3,5.);	52000
0	put(0,date9.);	01JAN1960

101

INPUT and PUT Functions

Data Set **Personnel**:

	Hired	First	Last	SSN
1	27MAR2003	Samatha	Jones	444444444
2	01SEP2006	Timothy	Peters	999999999

Hired is character.
SSN is numeric.

The following program is submitted:

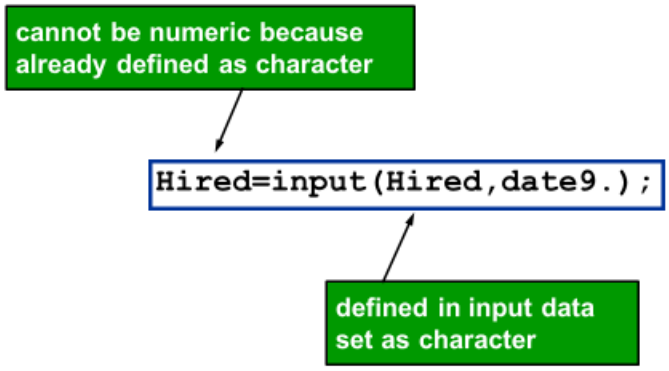
```
data NewPersonnel;
   set Personnel;
   Hired=input(Hired,date9.);
   SSN=put(SSN,9.);
run;
```

Yes or No Does this modified program create a numeric **Hired** and a character **SSN**?

104

INPUT and PUT Functions

After a variable type is established, it cannot be changed.



105

INPUT and PUT Functions

Solution:

```
data NewPersonnel;
   set Personnel
     (rename=(Hired=TempH SSN=TempS));
   Hired=input(TempH,date9.);
   SSN=put(TempS,9.);
   drop TempH TempS;
run;
```

- The RENAME= data set option changes the names of the original variables with unwanted data types.
- The DROP statement excludes the original variables with unwanted data types from the output SAS data set.

106

PUT Function: Example

The PUT function returns the value produced when *source* is written with *format*.

```
data conversion;
   NVar1=614;
   NVar2=55000;
   NVar3=366;
   CVar1=put(NVar1,3.);
   CVar2=put(NVar2,dollar7.);
   CVar3=put(NVar3,date9.);
run;
```

CharVar=**PUT**(*source*,*format*);

146 p205d21

INPUT Function

The INPUT function returns the value produced when the source is read with a specified informat.

NumVar=**INPUT**(*source*,*informat*);

```
data conversions;
   CVar1='32000';
   CVar2='32.000';
   CVar3='03may2008';
   CVar4='030508';
   NVar1=input(CVar1,5.);
   NVar2=input(CVar2,commax6.);
   NVar3=input(CVar3,date9.);
   NVar4=input(CVar4,ddmmyy6.);
run;
```

118 p205d16

INPUT Function

```
proc print data=conversions noobs;
run;
```

PROC PRINT Output

CVar1	CVar2	CVar3	CVar4	NVar1	NVar2	NVar3	NVar4
32000	32.000	03may2008	030508	32000	32000	17655	17655

120

Defining Character Variables

Set the length of the variable **Freq** to avoid truncation.

```
data work.bonus;
   set orion.sales;
   length Freq $ 12;
   if Country='US' then do;
      Bonus=500;
      Freq='Once a Year';
   end;
   else if Country='AU' then do;
      Bonus=300;
      Freq='Twice a Year';
   end;
run;
```

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

- It is a good practice to use a LENGTH statement any time you create a new character variable.

19 p109d08

9. Which of the following determines the length of a new variable at compile time?

   a. INPUT statement

   b. assignment statement

   c. LENGTH statement

   d. all of the above

42

DO Loops

The *DO loop* executes statements between DO and END repetitively based on the value of an index variable.

```
data training;
   miles=30;
   do weeks = 1 to 6 by 1;
      miles+miles*0.04;
   end;
run;
```

- Sally ran 30 miles per week.
- She plans to increase her mileage by 4% each week for six weeks.

108

Index Variable

The *index variable* names a variable whose value governs execution of the DO group.

index variable

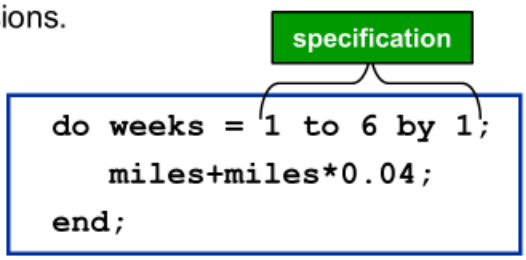
```
do weeks = 1 to 6 by 1;
  miles+miles*0.04;
end;
```

- The index variable argument is required.
- Unless you specify to drop it, the index variable is included in the data set that is being created.

109

Specification

The *specification* denotes an expression or a series of expressions.

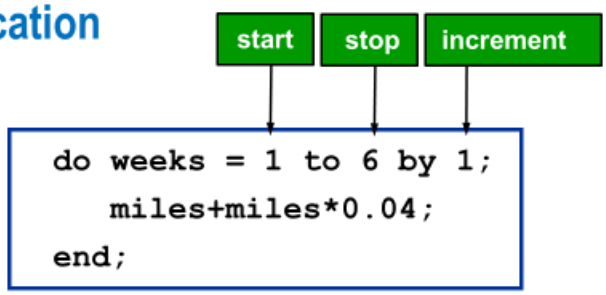


Possible specifications:

- ***start* TO *stop* BY *increment***
- ***start1*, *start2*, ...**
- **WHILE(*expression*)**
- **UNTIL(*expression*)**

110

Specification



- *start* specifies the initial value of the index variable.
- *stop* is an optional value that specifies the ending value of the index variable.
- *increment* is an optional value that specifies a positive or negative number to control the incrementing of the index variable.

  If no increment is specified, the index variable is increased by 1.

111

DO Loop Output

The following program is submitted:

```
data training;
   miles=30;
   do weeks = 1 to 6;
      miles+miles*0.04;
   end;
run;
proc print data=training noobs;
run;
```

implicit output

What is the result?

A.

miles	weeks
31.2000	1
32.4480	2
33.7459	3
35.0958	4
36.4996	5
37.9596	6

B.

miles	weeks
37.9596	6

C.

miles	weeks
37.9596	7

115

DO Loop Output

miles	weeks
37.9596	7

```
data training;
   miles=30;
   do weeks = 1 to 6;
      miles+miles*0.04;
   end;
   output;
run;
```

miles	weeks
31.2000	1
32.4480	2
33.7459	3
35.0958	4
36.4996	5
37.9596	6

```
data training;
   miles=30;
   do weeks = 1 to 6;
      miles+miles*0.04;
      output;
   end;
run;
```

miles	weeks
37.9596	6

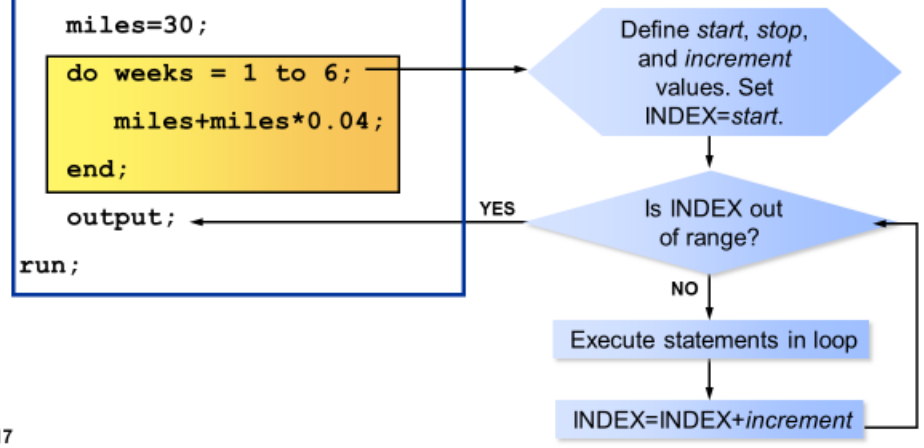
```
data training;
   miles=30;
   do weeks = 1 to 6;
     miles+miles*0.04;
     if weeks=6 then output;
   end;
run;
```

116

DO Loop Output

The value of increment is evaluated before the execution of the loop.

```
data training;
   miles=30;
   do weeks = 1 to 6;
      miles+miles*0.04;
   end;
   output;
run;
```



117

DO Loop with SET Statement

```
data training;
   set runners;
   do weeks=1 to 6;
     miles+miles*pct;
     output;
   end;
run;
```

VIEWTABLE: Work.Runners

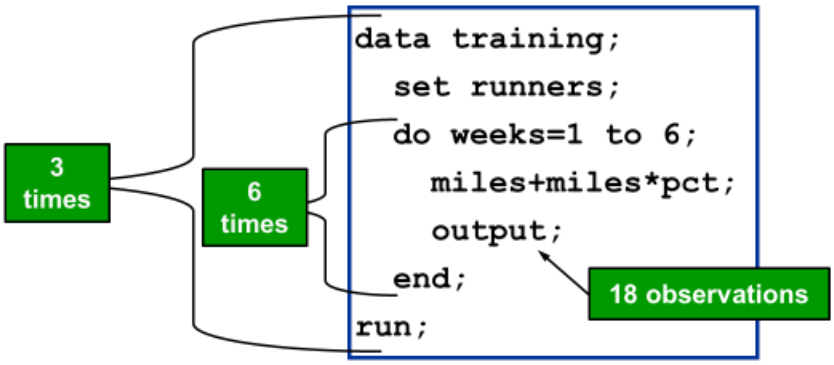
	name	miles	pct
1	Jill	24	0.05
2	Ray	28	0.04
3	Mark	30	0.05

1. How many times does SAS loop through the DATA step? 3
2. How many times does SAS loop through the DO loop per each DATA step iteration? 6
3. How many observations are created? 18

119

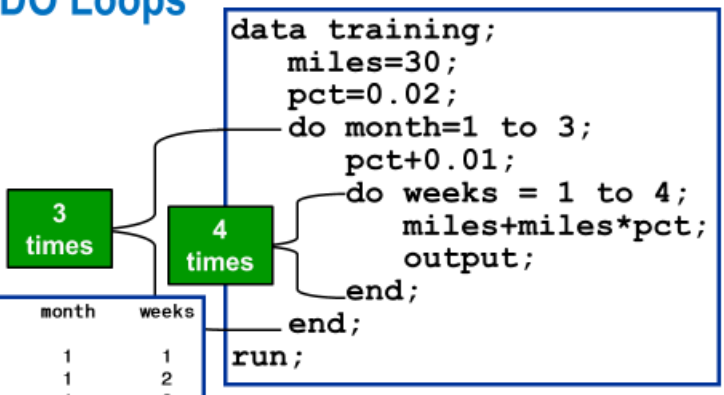
DO Loop with SET Statement

Three observations are in the data set **work.runners**.



120

Nested DO Loops



miles	pct	month	weeks
30.9000	0.03	1	1
31.8270	0.03	1	2
32.7818	0.03	1	3
33.7653	0.03	1	4
35.1159	0.04	2	1
36.5205	0.04	2	2
37.9813	0.04	2	3
39.5006	0.04	2	4
41.4756	0.05	3	1
43.5494	0.05	3	2
45.7269	0.05	3	3
48.0132	0.05	3	4

12 observations

124

Additional Specifications

The specification in the DO statement can be a series of items separated by commas.

- The items can be either all numeric or all character constants, or might be variables.
- Character constants must be enclosed in quotation marks.
- The DO group is executed once for each value in the list.

Examples:

- `do month = 'JAN','FEB','MAR';`
- `do count = 2,3,5,7,11,13,17;`
- `do i = var1, var2, var3;`
- `do date = '01JAN2007'd, '25APR2007'd;`

127

Additional Specifications

```
data training;
   miles=30;
   do date=today()+30, today()+100;
      miles+miles*0.10;
      output;
   end;
run;
```

miles	date
33.0	27MAY2007
36.3	05AUG2007

- Sally runs 30 miles per week.
- Thirty days from now she plans to increase her mileage by 10%.
- One hundred days from now she plans to increase her mileage again by 10%.

128

Conditional Specifications

The DO statement can execute statements repetitively while a condition is true or until a condition is true.

DO WHILE(*expression*)

- executes while a condition is true
- is evaluated at the ***top*** of the loop
- does not execute if the expression is false the first time that it is evaluated.

DO UNTIL(*expression*)

- executes until a condition is true
- is evaluated at the ***bottom*** of the loop
- is executed at least once.

129

Conditional Specifications

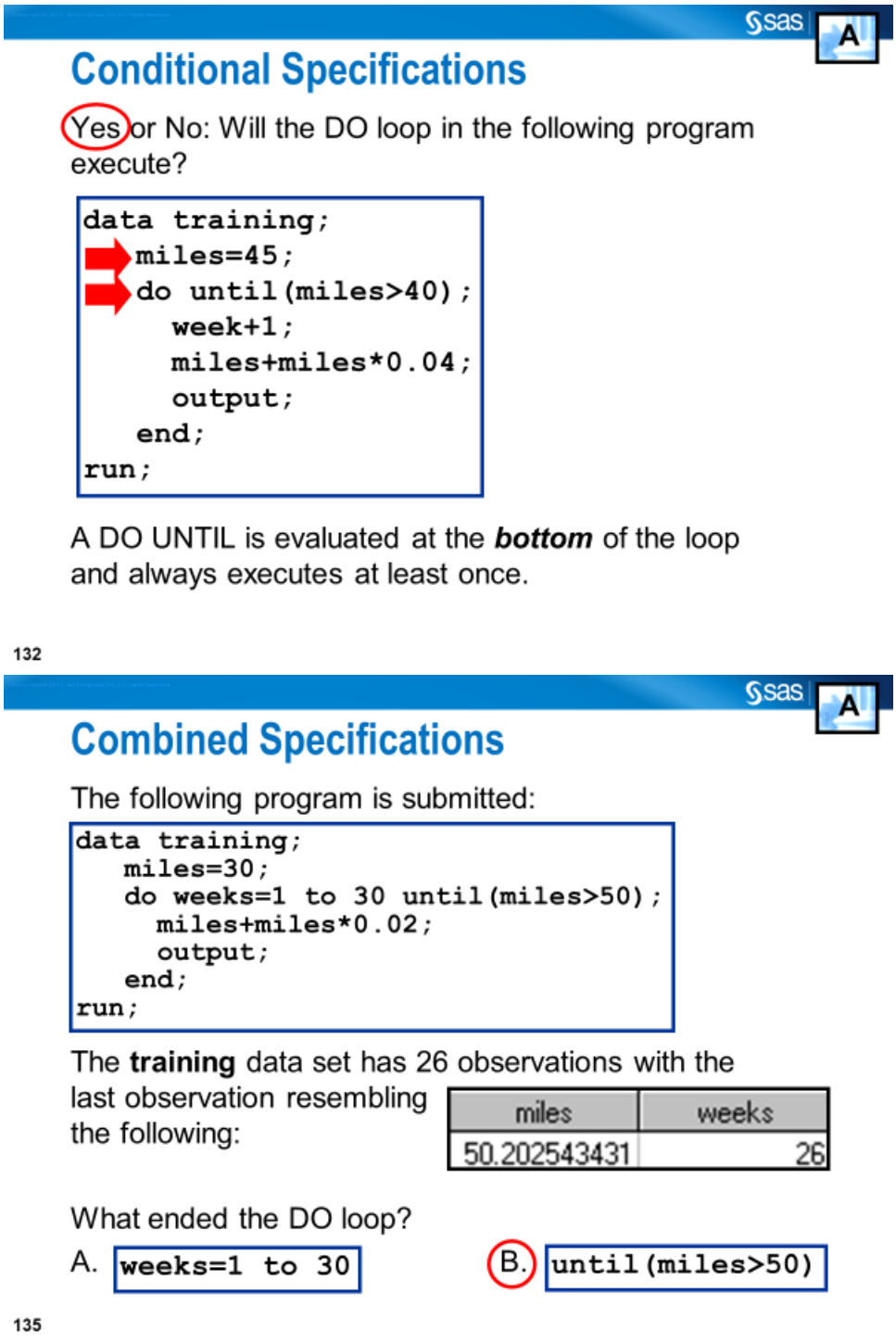
```
data training;
   miles=30;
   do while(miles<=50);
      week+1;
      miles+miles*0.04;
      output;
   end;
run;
```

```
data training;
   miles=30;
   do until(miles>50);
      week+1;
      miles+miles*0.04;
      output;
   end;
run;
```

	miles	week
1	31.2	1
2	32.45	2
3	33.75	3
4	35.1	4
5	36.5	5
6	37.96	6
7	39.48	7
8	41.06	8
9	42.7	9
10	44.41	10
11	46.18	11
12	48.03	12
13	49.95	13
14	51.95	14

Sally plans to increase her mileage by 4% each week until she runs 50+ miles per week.

130



Conditional Specifications

Yes or No: Will the DO loop in the following program execute?

```
data training;
   miles=45;
   do until(miles>40);
      week+1;
      miles+miles*0.04;
      output;
    end;
run;
```

A DO UNTIL is evaluated at the ***bottom*** of the loop and always executes at least once.

132

Combined Specifications

The following program is submitted:

```
data training;
   miles=30;
   do weeks=1 to 30 until(miles>50);
     miles+miles*0.02;
     output;
   end;
run;
```

The **training** data set has 26 observations with the last observation resembling the following:

miles	weeks
50.202543431	26

What ended the DO loop?

A. `weeks=1 to 30`

B. `until(miles>50)`

135

Ex20_do_loop_xtra.sas

Array Processing

You can use arrays to simplify programs that do the following:

- perform repetitive calculations
- create many variables with the same attributes
- read data
- compare variables
- perform a table lookup

64

Arrays

An *array* is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array name.

- Arrays exists only for the duration of the current DATA step.
- Arrays are referenced by the array name and a subscript.
- The array name is not a variable.

An array is only a convenient way of temporarily identifying a group of variables. Arrays are often referenced in DO loops because more than one element in an array must be processed.

138

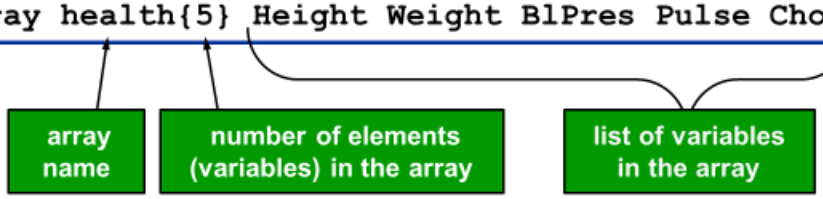
/

Defining an Array

An ARRAY statement defines elements of an array.

```
array run{4} week1-week4;
```

```
array health{5} Height Weight BlPres Pulse Chol;
```



- The number of elements must be enclosed in parentheses (), braces {}, or brackets [].
- Variables defined in a given array must be all character or all numeric.

140

Defining and Referencing an Array

VIEWTABLE: Work.Weekly

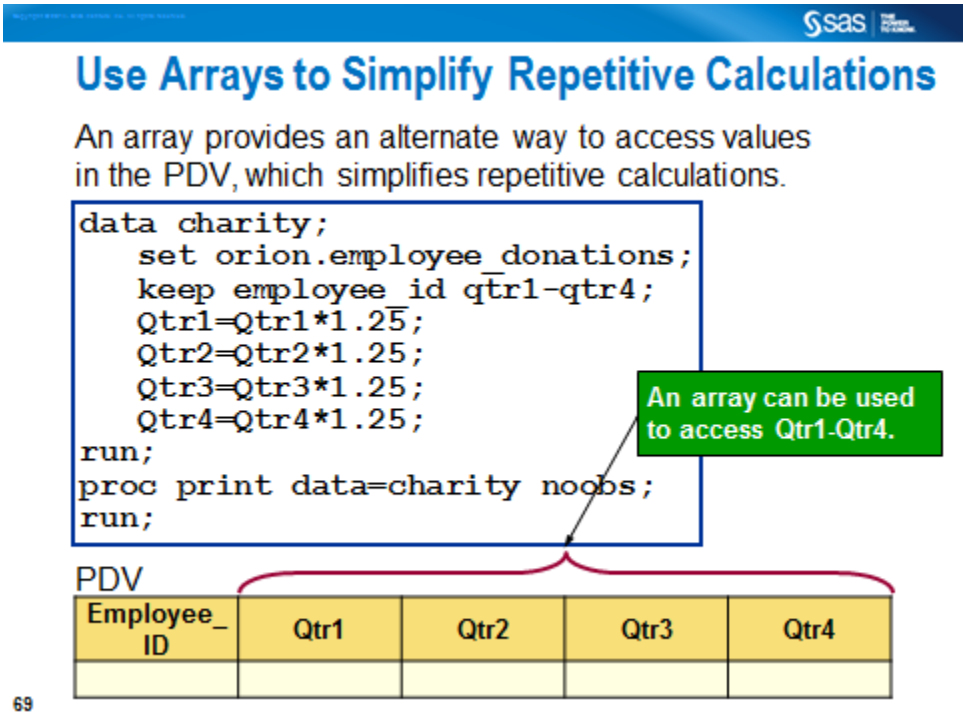
	name	week1	week2	week3	week4
1	Jack	25	32	48	33
2	Susan	10	12	10	10

```
data Increase;
   set Weekly;
   array run{4} week1 - week4;
   do week = 1 to 4;
      run{week} = run{week}*1.10;
   end;
   drop week;
run;
```

VIEWTABLE: Work.Increase

	name	week1	week2	week3	week4
1	Jack	27.5	35.2	52.8	36.3
2	Susan	11	13.2	11	11

142



Using a DO Loop to Process an Array

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{4} Qtr1-Qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

DO *index-variable*=1 TO *number-of-elements-in-array*;
<additional SAS statements>
END;

To reference an element, the index variable is often used as a subscript:

```
array-name{index-variable}
```

79 p207d11

First Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{4} Qtr1-Qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

when i=1

```
Contrib{1}=Contrib{1}*1.25;
```

```
Qtr1=Qtr1*1.25;
```

80 ...

Second Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{4} Qtr1-Qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

when i=2

```
Contrib{2}=Contrib{2}*1.25;
```

```
Qtr2=Qtr2*1.25;
```

81

...

Third Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{4} Qtr1-Qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

when i=3

```
Contrib{3}=Contrib{3}*1.25;
```

```
Qtr3=Qtr3*1.25;
```

82

...

Fourth Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{4} Qtr1-Qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

when i=4

```
Contrib{4}=Contrib{4}*1.25;
```

```
Qtr4=Qtr4*1.25;
```

83

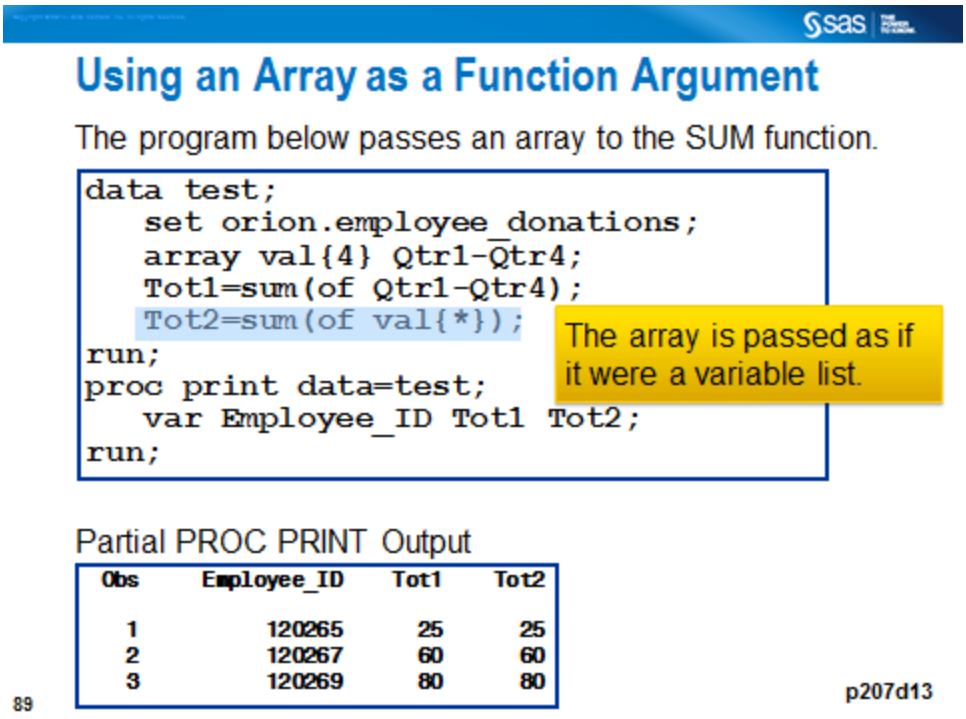
Output: Using a Do Loop to Process an Array

```
proc print data=charity noobs;
run;
```

Partial PROC PRINT Output

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	31.25
120267	18.75	18.75	18.75	18.75
120269	25.00	25.00	25.00	25.00
120270	25.00	12.50	6.25	.
120271	25.00	25.00	25.00	25.00
120272	12.50	12.50	12.50	12.50
120275	18.75	18.75	18.75	18.75
120660	31.25	31.25	31.25	31.25
120662	12.50	.	6.25	6.25

84



If you want to use all numeric (character or all) variables in the data set as elements in the array, you can use the special variable as elements

```
array  numvar[*] _NUMERIC_;
array  charvar[*] _CHARACTER_;
array  charvar[*] _ALL_;
```

DIM Function

The DIM function returns the number of elements in an array. This value is often used as the stop value in a DO loop.

```
data charity;
   set orion.employee_donations;
   keep Employee_ID Qtr1-Qtr4;
   array Contrib{*} qtr:;
   do i=1 to dim(Contrib);
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

DIM(*array_name*)

90 p207d12

Using an Array to Create Numeric Variables

An ARRAY statement can be used to create new variables in the program data vector.

```
array Pct{4} Pct1-Pct4;
```

If **Pct1** through **Pct4** do not exist in the PDV, they are created.

This statement produces the same results:

```
array Pct{4};
```

PDV

Pct1 N 8	Pct2 N 8	Pct3 N 8	Pct4 N 8

91

Acknowledgments: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts.

Using an Array to Create Character Variables

Define an array named **Month** to create six variables to hold character values with a length of 10.

```
array Month{6} $ 10;
```

PDV

Month1 $ 10	Month2 $ 10	Month3 $ 10	Month4 $ 10	Month5 $ 10	Month6 $ 10

92

Creating Variables with Arrays

```
data percent(drop=i);
   set orion.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Percent{4};
   Total=sum(of contrib{*});
   do i=1 to 4;
      percent{i}=contrib{i}/total;
   end;
run;
```

The second ARRAY statement creates four numeric variables: **Percent1**, **Percent2**, **Percent3**, and **Percent4**.

94 p207d14

Creating Variables with Arrays

```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```

The **Contrib** array refers to existing variables. The **Diff** array creates three variables: **Diff1**, **Diff2**, and **Diff3**.

Defining and Referencing an Array

```
data Stats2(drop=i);
   set Stats;
   count=0;
   array _________ height weight blpres pulse chol;
   do i = 1 to 5;
      if health{i} = 'AboveAve' then count+1;
   end;
run;
```

VIEWTABLE: Work.Stats2

	name	height	weight	blpres	pulse	chol	count
1	Jana	Ave	Ave	Ave	Ave	Ave	0
2	Tyler	Ave	AboveAve	AboveAve	Ave	AboveAve	3
3	William	Ave	Ave	Ave	Ave	AboveAve	1

Based on the program and the results of the final data set, what should be in the blank space?

A. `i{5}`
B. `count{5}`
C. `height{5}`
D. `health{5}`

144

Defining and Referencing an Array

When an asterisk is used to specify the number of elements, SAS is to determine the subscript by counting the variables in the array.

```
data newprices;
   set sashelp.pricedata;
   array parray{*} price:;
   total=sum(of parray{*});
   do num = 1 to dim(parray);
      parray{num} = parray{num}*1.10;
   end;
run;
```

The DIM function in the iterative DO statement returns the number of elements in an array.

145

Exercise 5 Solution

```
data WeeklyRotate;
   set Weekly;
   array run{4} week1 - week4;
   do week = 1 to 4;
      miles = run{week};
      output;
   end;
   drop week1 - week4;
run;
```

147

Creating Numeric Variables with an Array

An array can be based on existing variables or new variables.

```
data WeeklyDiff(drop=week);
   set Weekly;
   array run{4} week1 - week4;
   array diff{3} diff21 diff32 diff43;


   do week = 1 to 3;
      diff{week} = run{week+1}-run{week};
   end;
run;
```

array based on existing variables

array based on new variables

	name	week1	week2	week3	week4	diff21	diff32	diff43
1	Jack	25	32	48	33	7	16	-15
2	Susan	10	12	10	10	2	-2	0

148

Creating Numeric Variables with an Array

SAS creates variable names by concatenating the array name and the numbers 1, 2, 3, . . . *n*.

```
data WeeklyPct(drop=week);
   set Weekly;
   total=sum(of week1-week4);
   array run{4} week1 - week4;
   array pctwk{4};
   do week = 1 to 4;
      pctwk{week} = run{week}/total;
   end;
   format pctwk1-pctwk4 percent6.;
run;
```

array based on existing variables

array based on new variables

	name	week1	week2	week3	week4	total	pctwk1	pctwk2	pctwk3	pctwk4
1	Jack	25	32	48	33	138	18%	23%	35%	24%
2	Susan	10	12	10	10	42	24%	29%	24%	24%

149

TEMPORARY Option

The _TEMPORARY_ option is used to create a list of temporary data elements.

```
data score(drop=x);
   infile 'raw-data-file';
   input name $ q1 $ q2 $ q3 $ q4 $ q5 $;
   array answer{5} q1-q5;
   array correct{5} $ 1 _temporary_
                    ('A','B','A','D','C');
   score=0;
   do x=1 to 5;
      if answer{x}=correct{x}
         then score+1;
   end;
run;
```

	name	q1	q2	q3	q4	q5	score
1	monica	A	C	B	D	C	3
2	ted	A	B	A	D	C	5
3	kelly	A	C	C	A	C	2

154

Some common forms of the ARRAY statement

```
ARRAY test [5] test1- test5;
ARRAY test [1:5] test1- test5;
ARRAY test [*] test1- test5;
ARRAY test[5];
```

Introduction

There is often a need to combine or look up data from multiple sources to create meaningful reports.

4

Introduction

When data sources do not share a common structure, a lookup table can match them.

1

Lookup Table Data

You can use several techniques to look up data.

Continent ID	Continent Name
91	North America
93	Europe
94	Africa
95	Asia
96	Australia/Pacific

6

IF-THEN/ELSE Statements

Lookup tables can be SAS programming statements.

```
data countryinfo;
   set orion.country;
   if ContinentID=91
      then Continent='North America';
   else if ContinentID=93
      then Continent='Europe';
   else if ContinentID=94
      then Continent='Africa';
   else if ContinentID=95
      then Continent='Asia';
   else if ContinentID=96
      then Continent='Australia/Pacific';
run;
```

7

DATA Step Merge

Lookup tables can be SAS data sets that are accessed during a DATA step merge.

```
data countryinfo;
   merge orion.country
         orion.Continent;
   by ContinentID;
run;
```

8

PROC SQL Join

Lookup tables can be SAS data sets that are accessed during a PROC SQL join.

```
proc sql;
   create table countryinfo as
      select * from
        orion.country a, orion.Continent b
        where a.ContinentID=b.ContinentID;
quit;
```

9

User-Defined Formats

Lookup tables can be user-defined formats that are accessed with a FORMAT statement or PUT function.

```
proc format;
   value ContName
      91='North America'   93='Europe'
      94='Africa'          95='Asia'
      96='Australia/Pacific';
run;

proc print data=orion.country;
   format ContinentID ContName.;
run;

data countryinfo;
   set orion.country;
   Continent=put(ContinentID,ContName.);
run;
```

10

Arrays

Lookup tables can be arrays.

```
data countryinfo;
   array ContName{91:96} $ 30 _temporary_
      ('North America',
         ' ',
      'Europe',
      'Africa',
      'Asia',
      'Australia/Pacific');
   set orion.country;
   Continent=ContName{ContinentID};
run;
```

11

Table Lookup Techniques

The technique that is used to perform a table lookup depends on the data.

Table Location	Technique	Lookup Table
DATA step	IF-THEN/ELSE statements	SAS programming statements
Disk	SQL join merge SET/SET KEY=	SAS data set
Memory	FORMAT statement PUT statement PUT function	user-defined format
	array reference	array
	FIND method	hash object

12

Reviewing SAS Arrays

An array is declared with an ARRAY statement. Array elements can be SAS variables or temporary data elements.

```
array ContName{91:96} $ 30 _temporary_
          ('North America',
           ' ',
           'Europe',
           'Africa',
           'Asia',
           'Australia/Pacific');
```

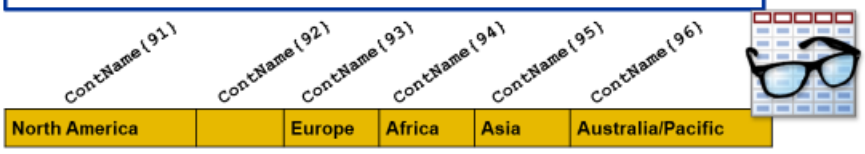
ARRAY *array-name* {*number-of-elements*} <$> <*length*> <*_temporary_*> <*list-of-variables*> <(*initial-values*)>;

24

Reviewing SAS Arrays

This ARRAY statement assigns an initial value to each corresponding element. SAS matches the elements and values by position, so the values must be listed in the order of the array elements.

```
array ContName{91:96} $ 30 _temporary_
          ('North America',
           ' ',
           'Europe',
           'Africa',
           'Asia',
           'Australia/Pacific');
```



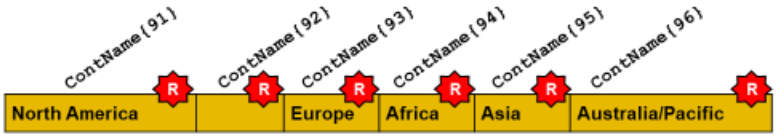
25

...

Reviewing SAS Arrays

When an initial value list is specified, all array elements behave as if they were named in a RETAIN statement.

```
array ContName{91:96} $ 30 _temporary_
          ('North America',
           ' ',
           'Europe',
           'Africa',
           'Asia',
           'Australia/Pacific');
```



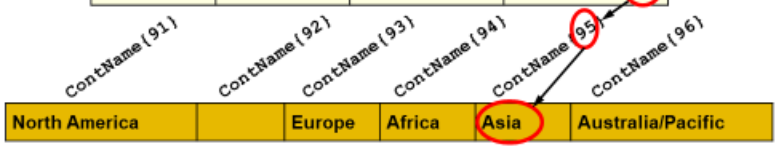
26

Example: Looking Up Continent Names Using an Array

A one-dimensional array can serve as a lookup table with the names of the continents.

Partial **orion.country**

Country	Country Name	Population	ContinentID
AU	Australia	20,000,000	96
CA	Canada	.	91
DE	Germany	80,000,000	93
IL	Israel	5,000,000	95
TR	Turkey	70,000,000	95



29

Example: Looking Up Continent Names Using an Array

```
data countryinfo;
   array ContName{91:96} $ 30 _temporary_
         ('North America',
          ' ',
          'Europe',
          'Africa',
          'Asia',
          'Australia/Pacific');
   set orion.country;
   Continent=ContName{ContinentID};
run;
```

30