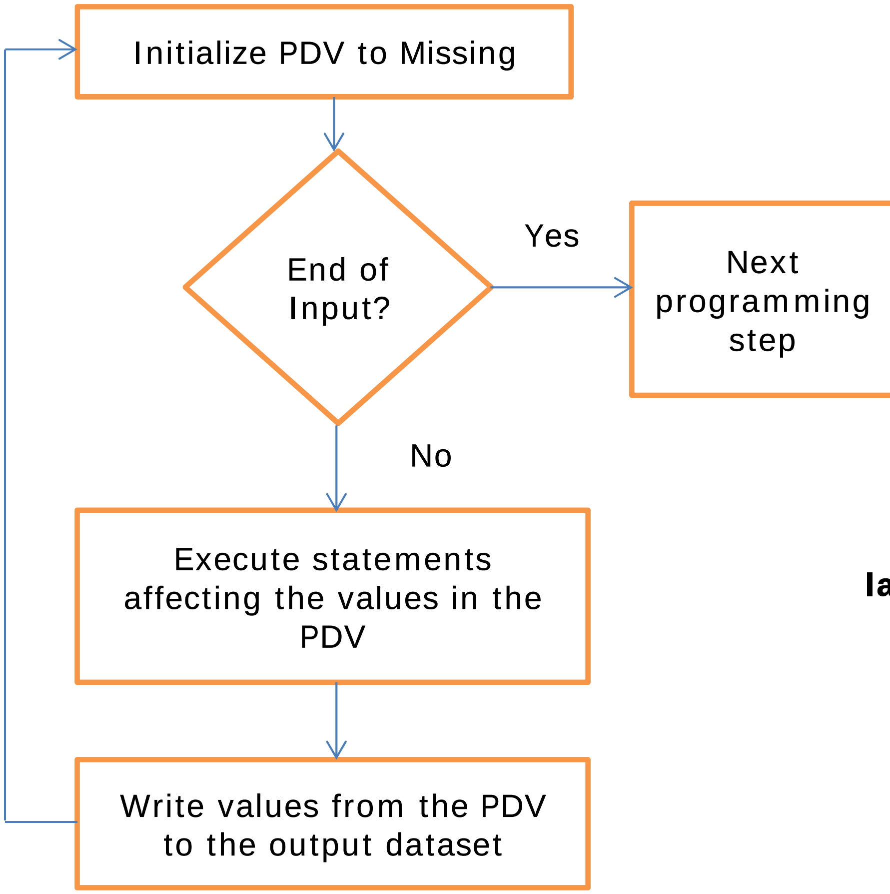
Chapter 4: The DATA Step – Part II

- RETAIN statement
- RENAME and LABEL statements
- Conditional execution
- Indicator variables
- Formats in a DATA step
- Creating user-defined formats, including PUT function
- DO groups
- Arrays
- Transformations involving missing values
- Sum statement

Declarative Statements

- These statements supply information to SAS during the compilation phase of DATA step processing.
- Declarative statements do the following:
    - Define and modify the actions that are subsequently taken during the execution phase.
    - Affect the composition and contents of the PDV and the new data set being created.
    - For example, the DROP and KEEP statements determine which of the variables in the PDV get output to the new data set.
- Declarative statements are "non-executable" and their placement in the DATA step usually is unimportant, although there are cases where the order of these statements does affect the outcome.
- Remember, the PDV is created during the compile phase by compiling the statements in the order in which the compiler comes to them.

The Compilation Phase of the DATA Step Creates:



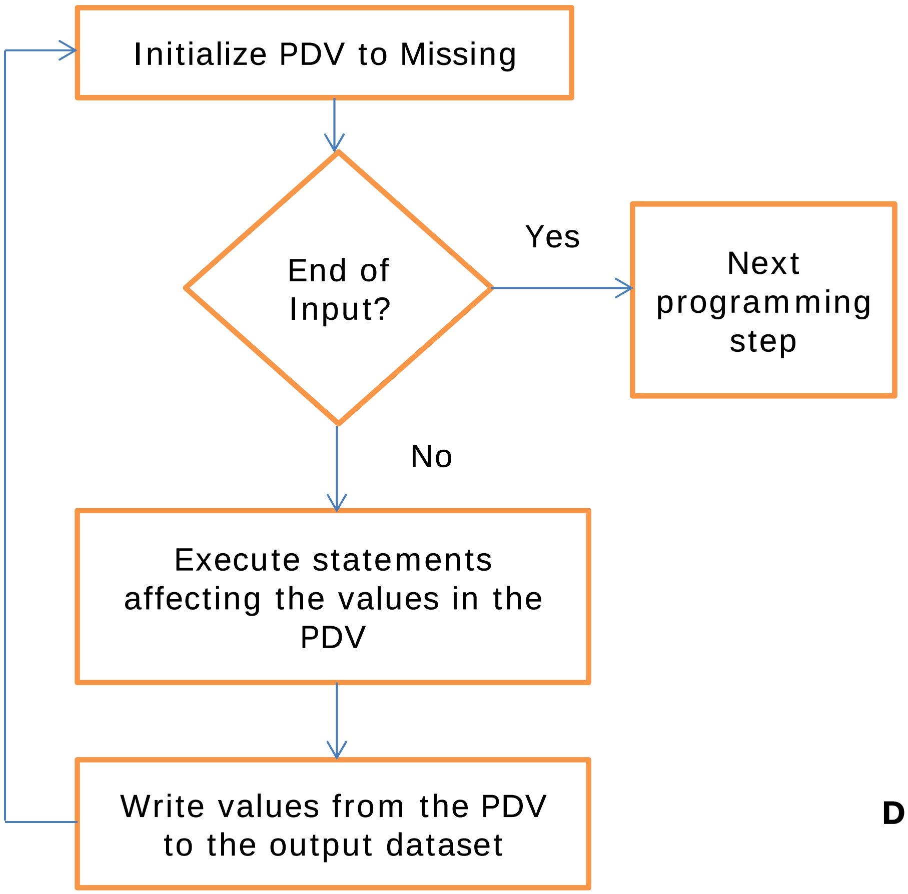
1. The machine language program

2. The Program Data Vector (PDV)

<u>Variable Attributes</u>
Name
Type
Length
Format
Label

3. The descriptor portion of the output dataset

The Following Declarative Statements Affect The Compilation Phase:



RETAIN

DROP, KEEP, RENAME

LENGTH, LABEL, FORMAT

<u>Variable Attributes</u>
Name
Type
Length
Format
Label

The RETAIN Statement

- The RETAIN statement lists those variables in the PDV that should not be initialized to missing at the beginning of each execution of the DATA step.
- Syntax: `RETAIN VARIABLE-LIST <INITIAL VALUE> ...;`

  where:

  - *VARIABLE-LIST* is a list of variable names to be exempt from being reset to missing.
  - *INITIAL VALUE* is the <optional> value placed in the PDV at compile time.
- Multiple RETAIN statements may be entered in the same DATA step.
- A single RETAIN statement may specify both numeric and character variables.
- If the first reference to a variable is in the RETAIN statement, SAS assumes it is numeric.
  - To indicate a character variable, provide an initial value of the proper length.
  - A better strategy is to define the variable in a preceding LENGTH statement.
- Only variables created by assignment and INPUT statements may be retained.
- Retaining a variable brought into the DATA step with a SET, MERGE, or UPDATE statement is not an error, just an action with no effect.
- Values of retained variables are held over from the last observation, but they can be changed with SET or assignment statements.
- If no initial value is given, character variables are initially blank and numeric variables are initially missing (.).

Figure 1: Computing Total Sums with a RETAIN Statement

```
proc print data= bios511.class; run;
```

Obs	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

```
data one;
 set bios511.class;
  cumht  = cumht  + ht;
  cumwt  = cumwt  + wt;
  cumage = cumage + age;
run;
proc print data=one; run;
```

Obs	NAME	SEX	AGE	HT	WT	cumht	cumwt	cumage
1	CHRISTIANSEN	M	37	71	195	.	.	.
2	HOSKING J	M	31	70	160	.	.	.
3	HELMS R	M	41	74	195	.	.	.
4	PIGGY M	F	.	48	.	.	.	.
5	FROG K	M	3	12	1	.	.	.
6	GONZO		14	25	45	.	.	.

```
data two;
 set bios511.class;
  retain cumht cumwt 0 cumage;
  cumht  = cumht  + ht;
  cumwt  = cumwt  + wt;
  cumage = cumage + age;
run;
proc print data=two; run;
```

Obs	NAME	SEX	AGE	HT	WT	cumht	cumwt	cumage
1	CHRISTIANSEN	M	37	71	195	71	195	.
2	HOSKING J	M	31	70	160	141	355	.
3	HELMS R	M	41	74	195	215	550	.
4	PIGGY M	F	.	48	.	263	.	.
5	FROG K	M	3	12	1	275	.	.
6	GONZO		14	25	45	300	.	.

Figure 2: Computing Counts with a RETAIN Statement

```
data one;
 set bios511.class;
  retain count 0 nmales 0;

  count  = count   + 1;
  nmales = nmales  + (sex='M');

 keep name sex count nmales;
run;

proc print data=one; run;
```

Obs	NAME	SEX	count	nmales
1	CHRISTIANSEN	M	1	1
2	HOSKING J	M	2	2
3	HELMS R	M	3	3
4	PIGGY M	F	4	3
5	FROG K	M	5	4
6	GONZO		6	4

The RENAME Statement

- The RENAME statement changes the name of a variable between the PDV and the output data set.
- Syntax: `RENAME oldname1=newname1 oldname2=newname2 ...;`

Figure 3: Renaming variables with a RENAME Statement

```
proc print data = bios511.class; run;

data class;
 set bios511.class;
 rename name=lastname ht=height;
 if ht > 20;
run;

proc print data = class; run;
```

PROC PRINT of “bios511.class”

Obs	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

PROC PRINT of “work.class”

Obs	lastname	SEX	AGE	height	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	GONZO		14	25	45

- The names of the variables in the input dataset bios511.CLASS are:

NAME	SEX	AGE	HT	WT

- The names of the variables in the output dataset B are:

Lastname	SEX	AGE	height	WT

Using Labels in a DATA Step

- The LABEL statement is used during the DATA step to add variable labels of up to 256 characters each to the descriptor section of the output data set being created.
- Syntax: `LABEL VARIABLE = '256 character label' ... ;`
- Labels containing special characters such as a double quote or a semi-colon must be enclosed in single quotes.
- Labels containing single quotes must be enclosed in double quotes.
- For example, to assign the label "Subject's Name" to the variable NAME, one would do the following:

```
LABEL name = "Subject's Name";
```

- Outside of these cases, labels can be enclosed in single or double quotes.
- Existing variable labels from the input data set are included on the output data set, unless they are modified by a LABEL statement.
- Good labels should be used when you create a data set to make the data set “self-documenting”.
- However, self-documenting labels aren’t always optimal for display purposes, so you can always do one of the following:
  - Submit `OPTIONS NOLABEL;` to ask SAS to stop using labels in your output and then submit `OPTIONS LABEL;` to ask SAS to resume its normal use of labels.
  - Assign temporary labels in a PROC step.

Figure 4: Using LABEL Statements

<table>
<tr>
<td>

```
data one;
 set bios511.class;
  label sex =
   "Gender (M=Male, F=Female");
run;

proc print data=one(obs=2) label;
 var Name Sex Age;
run;
```

</td>
<td>

Obs	NAME	Gender (M=Male,F=Female,Missing=Gender Not Provided)	AGE
1	CHRISTIANSEN	M	37
2	HOSKING J	M	31

</td>
</tr>
<tr>
<td>

```
option nolabel;
proc print data=one(obs=2) label;
 var Name Sex Age;
run;
option label;
```

</td>
<td>

Obs	NAME	SEX	AGE
1	CHRISTIANSEN	M	37
2	HOSKING J	M	31

</td>
</tr>
<tr>
<td>

```
proc print data=one(obs=2) label;
 var Name Sex Age;
 label Sex = 'Gender';
run;
```

</td>
<td>

Obs	NAME	Gender	AGE
1	CHRISTIANSEN	M	37
2	HOSKING J	M	31

</td>
</tr>
</table>

Conditional Execution of Statements

- Up to this point, the statements in the DATA step have been executed sequentially for each observation processed.
- The ability to execute or not execute a statement based on whether or not some condition is met – that is, to make logical decisions based on data values - is one of the most powerful features of a computer.
- We have already discussed a very specialized conditional statement, the subsetting IF statement, which is used to control whether or not observations are added to the output data set.
- The general form of the IF statement is:

```
IF expression THEN statement1;
          ELSE statement2;
```

  where *expression* is any valid SAS expression, and statements 1 and 2 are any executable SAS statements.

- If the expression is "true" (non-zero and non-missing) then statement 1 is executed. If expression is "false" (zero or missing) then statement 2 is executed.
- The expression is usually a comparison expression (e.g., x<4), in which case the expression has a value of 1 for true and 0 for false.
- Arithmetic expressions (e.g., y + z) are also valid.
- The ELSE statement is optional. If it is not used and the expression is false, control is transferred to the next statement.

Figure 5: Understanding IF/THEN/ELSE processing

Not Functional

```
data class;
 set bios511.class;

 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then sexn = 1;
                   sexL = 'Male';
   else sexn = 0;
        sexL = 'Female';


run;
```

```
Log - (Untitled)
data class;
 set bios511.class;
 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then sexn    = 1;
                   sexL = 'Male';
  else sexn    = 0;
  ----
  160
ERROR 160-185: No matching IF-THEN clause.

        sexL = 'Female';
run;

NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.CLASS may be incomplete.  When this
         step was stopped there were 0 observations and 7
         variables.
WARNING: Data set WORK.CLASS was not replaced because this step
         was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Functions / Correct

```
data class;
 set bios511.class;

 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then sexn = 1;
   else sexn = 0;

 if sex = 'M' then sexL = "Male";
   else sexL = "Female";

run;
```

```
data class;
 set bios511.class;
 where sex in ("M","F");

 length sexL $6;

 if sex = 'M' then sexn = 1;
   else sexn = 0;

 if sex = 'M' then sexL = "Male";
   else sexL = "Female";
run;

NOTE: There were 5 observations read from the data set
      BIOS511.CLASS.
      WHERE sex in ('F', 'M');
NOTE: The data set WORK.CLASS has 5 observations and 7
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Figure 6: IF/ THEN/ ELSE RETAIN Example

```
data one;
 set bios511.class;

 retain nMales nFemales nMissing 0;

      if sex = 'M' then nMales   = nMales + 1;
 else if sex = 'F' then nFemales = nFemales + 1;
 else                   nMissing = nMissing + 1;

run;

proc print data=one; run;
```

Obs	NAME	SEX	AGE	HT	WT	nMales	nFemales	nMissing
1	CHRISTIANSEN	M	37	71	195	1	0	0
2	HOSKING J	M	31	70	160	2	0	0
3	HELMS R	M	41	74	195	3	0	0
4	PIGGY M	F	.	48	.	3	1	0
5	FROG K	M	3	12	1	4	1	0
6	GONZO		14	25	45	4	1	1

Figure 7: IF/ THEN/ ELSE RETAIN Example + Subsetting IF

```
data two;
 set bios511.class;

 retain nMales nFemales nMissing 0;

      if sex = 'M' then nMales   = nMales + 1;
 else if sex = 'F' then nFemales = nFemales + 1;
 else                   nMissing = nMissing + 1;

 if _n_ = 6; ** there is a better way;

 pMales   = nMales / _n_ * 100;
 pFemales = nFemales / _n_ * 100;
 pMissing = nMissing / _n_ * 100;

 keep nMales pMales nFemales pFemales nMissing pMissing;
run;

proc print data=two; run;
```

Obs	nMales	nFemales	nMissing	pMales	pFemales	pMissing
1	4	1	1	66.6667	16.6667	16.6667

DO/END Statements

- The DO and END statements define the beginning and end of a group of statements called a DO group. The DO group can be used within IF-THEN/ELSE statements to conditionally execute groups of statements.
- Execution of a DO statement specifies that all statements between the DO and its matching END statement are to be executed.
- Syntax:

```
DO;
 Statement1;
    .
    .
    .
 StatementN;
END;
```

Figure 8: DO Group Example

Functions / Correct

```
data class;
 set bios511.class;

 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then do;
   sexn = 1;
   sexL = 'Male';
 end;
 else do;
   sexn = 0;
   sexL = 'Female';
 end;
run;
```

```
Log - (Untitled)
data class;
 set bios511.class;

 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then do;
   sexn = 1;
   sexL = 'Male';
 end;
 else do;
   sexn = 0;
   sexL = 'Female';
 end;
run;

NOTE: There were 5 observations read from the data set
      BIOS511.CLASS.
      WHERE sex in ('F', 'M');
NOTE: The data set WORK.CLASS has 5 observations and 7
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

Functions / Correct

```
data class;
 set bios511.class;

 where sex in ("M","F");
 length sexL $6;

 if sex = 'M' then sexn = 1;
   else sexn = 0;


if sex = 'M' then sexL = "Male";
   else sexL = "Female";


run;
```

```

data class;
 set bios511.class;
 where sex in ("M","F");

 length sexL $6;

 if sex = 'M' then sexn = 1;
   else sexn = 0;

 if sex = 'M' then sexL = "Male";
   else sexL = "Female";
run;

NOTE: There were 5 observations read from the data set
      BIOS511.CLASS.
      WHERE sex in ('F', 'M');
NOTE: The data set WORK.CLASS has 5 observations and 7
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Creating & Using Indicator Variables

- An indicator variable is numeric and takes the values of 1 or 0 depending on whether a condition is TRUE or FALSE.
- It is often desirable to create indicator variables from one or more input variables.
- Indicator variables can be easily created using IF/THEN logic or Boolean expressions.
- When creating indicator variables, ALWAYS make sure that you evaluate the accuracy of your code in the presence of missing data.

Figure 9: Indicator Variables / Missing Data Example

```
proc print data = bios511.fitness(obs=10);
run;

data fitness;
 set bios511.fitness;

 ageCat1 = (age>40);  ** could be OK in some cases;

      if age>40 then ageCat2 = 1;
 else if age>.  then ageCat2 = 0;

 ageCat3 = ageCat1;
 if age = . then ageCat3 = .;

run;
proc print data = fitness(obs=10);
 var Teacher age ageCat:;
run;
```

Obs	Teacher	Age	Sex	Heart	Exer	Aero
1	Yang	28	M	86	2	36.6
2	Yang	41	M	76	3	26.7
3	Yang	30	M	78	2	33.8
4	Yang	39	F	90	1	13.6
5	Yang	28	M	96	1	33.0
6	Yang	26	M	74	2	42.7
7	Yang	.	F	66	4	36.1
8	Yang	48	F	72	2	22.6
9	Yang	31	M	60	3	44.1
10	Reed	28	F	84	2	22.1

Obs	Teacher	Age	ageCat1	ageCat2	ageCat3
1	Yang	28	0	0	0
2	Yang	41	1	1	1
3	Yang	30	0	0	0
4	Yang	39	0	0	0
5	Yang	28	0	0	0
6	Yang	26	0	0	0
7	Yang	.	0	.	.
8	Yang	48	1	1	1
9	Yang	31	0	0	0
10	Reed	28	0	0	0

- Indicator variables have many uses, and here is an especially valuable one for this class.
- The mean value of an indicator variable is the proportion of observations with the indicated state.
- For example, consider a data set of 20 people that includes an indicator variable for the disease diabetes.
    - Suppose 5 people have diabetes, therefore the value of the indicator variable is 1 for them.
    - The remaining 15 people do not have diabetes, therefore the value of the indicator variable is 0 for them.
    - The mean value of the diabetes indicator is therefore: ((5*1)+(15*0))/20 = 5/20 = .25, which is the proportion of people in the data set with diabetes.
- If you multiply an indicator variable by 100 so that the values become 0 and 100, then the mean of the variable is the percentage (value between 0 and 100) of observations with the indicated state.

Figure 10: Computing the Mean of an Indicator Variable as a Proportion

```
proc sort data = bios511.clin4new out = clin1;
 by sex;
run;

proc freq data= clin1 noprint;
   by sex;
   tables highbp / out = freq_output;
run;

proc print data = freq_output; run;
```

Obs	SEX	HIGHBP	COUNT	PERCENT
1	1	0	173	78.2805
2	1	1	48	21.7195
3	2	0	219	78.7770
4	2	1	59	21.2230

```
data clin2;
    set bios511.clin4new;
    highbp100 = highbp*100;
run;

proc means data=clin2 noprint nway;
 class sex;
 var highbp highbp100;
 output out = means_output
      n(highbp)     = N
     sum(highbp)    = Count
   mean(highbp100) = Percent;
run;

proc print data = means_output; run;
```

Obs	SEX	_TYPE_	_FREQ_	N	Count	Percent
1	1	1	221	221	48	21.7195
2	2	1	278	278	59	21.2230

Controlling Printing of Values Using Formats

- A SAS format is an instruction that SAS uses to write data values.
- For example, if you had the number 587, you could ask SAS to apply the WORDS. format and display the value as "five hundred eighty-seven".
- Formats are used to do the following:
  - Control the written appearance (or display) of data values to make your SAS output more readable.
  - Print numeric values as character values (for example, print 1 as Male and 2 as Female).
  - Print one character string as another character string (for example, print YES as OUI).
  - Group data values together for analysis.
- SAS format names are referenced using the following syntax:

<$>FORMAT-NAME<w>.<d>

where:

$: Indicates a character format; no $ indicates a numeric format (character formats are applied to character variables, numeric formats to numeric variables).
FORMAT-NAME : Gives the name of the format to be applied. Can be either a format supplied by SAS or a user-defined format created with PROC FORMAT.
w : (optional) Gives the number of characters to be used in the display; if omitted, SAS makes a reasonable choice.
d : (optional) For numeric formats, how many of the w are to be to the right of the decimal place.

- The name of a format that is being supplied <u>always</u> contains a period (.) as part of the name.
- SAS issues an error message if you try to use a character format with a numeric variable or vice versa.

- Most formats support widths and alignments.
  - Width refers to the number of columns used to display the values.
  - Alignment refers to how SAS behaves when a value's formatted length is less than the specified width.
  - Usually, numeric formats right-align and character formats left-align.
  - This means that if there are leftover columns, numbers are shifted flush right in the columns and characters are shifted flush left in the columns.
- If a value is displayed as a series of asterisks (***), the applied format width was too narrow for the value.
- Formats are applied to variables using a FORMAT statement which has the following syntax:

```
FORMAT VARIABLE-LIST FORMAT-NAME. ... ;
```

where

VARIABLE-LIST is a collection of SAS variables having the same type
FORMAT-NAME is the name of a SAS format

- Applying a format does not change the "internal" values of a variable. It only changes the way the values are displayed.
- As with LABEL statements, FORMAT statements can attach formats to variables permanently if used in the DATA step and temporarily if used in a PROC step.

Figure 11: Using the FORMAT statement

```
proc print data=bios511.sales(obs=2); run;

proc contents data = bios511.sales; run;
```

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY
1	SHOES	CLEVER	99.95	41.21	TUE	3
2	SHOES	AGILE	95.00	40.49	WED	4

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	CLERK	Char	6
4	COST	Num	8
6	DAY	Num	8
1	DEPT	Char	5
3	PRICE	Num	8
5	WEEKDAY	Char	3

```
proc print data=bios511.sales(obs=2);
      format cost price dollar9.2;
run;
```

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY
1	SHOES	CLEVER	$99.95	$41.21	TUE	3
2	SHOES	AGILE	$95.00	$40.49	WED	4

```
** permanently assign format;
data sales;
 set bios511.sales(obs=2);
 format cost price dollar9.2;
run;

proc print data=sales; run;

proc contents data = sales; run;
```

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY
1	SHOES	CLEVER	$99.95	$41.21	TUE	3
2	SHOES	AGILE	$95.00	$40.49	WED	4

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
2	CLERK	Char	6	
4	COST	Num	8	DOLLAR9.2
6	DAY	Num	8	
1	DEPT	Char	5	
3	PRICE	Num	8	DOLLAR9.2
5	WEEKDAY	Char	3	

```
data sales;
 set bios511.sales(obs=2);
 format cost price dollar9.2;
run;

** clear format for printing;
proc print data=sales;
 format cost price;
run;
```

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY
1	SHOES	CLEVER	99.95	41.21	TUE	3
2	SHOES	AGILE	95.00	40.49	WED	4

Selected SAS-Supplied Formats

Format	Definition	Width range (default)	Alignm ent	Input data	Format applied	Result
Character						

$w.	Writes standard character data – does not trim leading blanks (same as $CHARw.)	1-32767 (length of variable or 1)	left	Jacqueline
				  Tar Heel
				squash
					$9.
					$9.
					$9.
						Jacquelin
						  Tar Heel
						squash

Numeric						
BESTw.	SAS chooses best notation	1-32 (12)	right	3400001	BEST6. BEST8.	3.4E6 3400001
COMMAw.d	Uses commas and decimal points	2-32 (6)	right	3400001	COMMA9. COMMA12.2	3,400,001 3,400,001.00
DOLLARw.d	Uses dollar signs, commas, and decimal points	2-32 (6)	right	3400001	DOLLAR10. DOLLAR13.2	$3,400,001 $3,400,001.00
Ew.	Writes scientific notation	7-32 (12)	right	3400001	E7.	3.4E+06
w.d	Writes standard numeric data	1-32 (none)	right	52.107	6.3 5.2	52.107 52.11
WORDSw.	Writes numeric values as words	5-32767 (10)	left	111	WORDS20.	one hundred eleven

Format	**Definition**	**Width range (default)**	**Alignm ent**	**Input data**	**Format applied**	**Result**
Date						
DATEw.	Writes date values in form *ddmmmyy* or *ddmmmyyyy*	5-9 (7)	right	15597	DATE7. DATE9.	14SEP02 14SEP2002
DATETIMEw.d	Write SAS datetime values in form *ddmmmyy:hh:mm:ss.ss*	7-40 (16)	right	11107587	DATETIME13. DATETIME18.1	01JAN60:04:19 01JAN60:04:19:57.0
DAYw.	Writes day of month from a SAS date value	3-32 (2)	right	15597	DAY2. DAY5.	14 14
MMDDYYw.	Writes SAS date values in form *mmddyy* or *mmddyyyy*	2-10 (8)	right	15597	MMDDYY6. MMDDYY8. MMDDYY10.	091402 09/14/02 09/14/2002
EURDFDDw.	Writes SAS date values in form *dd.mm.yy*	2-10 (8)	right	15597	EURDFDD6. EURDFDD8. EURDFDD10.	140902 14.09.02 14.09.2002
TIMEw.d	Writes SAS time values in form *hh:mm:ss.ss*	2-20 (8)	right	27301	TIME8. TIME11.2	7:35:01 7:35:01.00

Format	Definition	Width range (default)	Alignment	Input data	Format applied	Result
WEEKDATEw.	Writes SAS date values in form *day-of-week, month-name dd, yy* or *yyyy*	3-37 (29)	right	15597	WEEKDATE15. WEEKDATE29.	Sat, Sep 14, 02 Saturday, September 14, 2002
WORDDATEw.	Writes SAS date values in form *month-name dd, yyyy*	3-32 (18)	right	15597	WORDDATE12. WORDDATE18.	Sep 14, 2002 September 14, 2002

Figure 12: Example with Insufficient Format Width

```
proc print data=bios511.class;
 format ht wt 2.0 ;
run;
```

Obs	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	**
2	HOSKING J	M	31	70	**
3	HELMS R	M	41	74	**
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

```
proc print data=bios511.class;
 format ht wt best. ;
run;
```

Obs	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

Creating User-Defined Formats

- You can create your own formats using the FORMAT procedure.
- These formats can be used to assign value labels for variables in either DATA or PROC steps.
- User-defined formats can also be used to recode variables in a DATA step or to collapse variable categories in a PROC step.
- There are two types of user-defined formats.
    - VALUE formats convert one or more user-specified values into a single character string (for display only).
    - PICTURE formats (not covered in these notes) specify a template for how to print a number or range of numbers.
- Syntax:

```
PROC FORMAT;
VALUE format1 range1 = "LABEL1"
                      .
                        .
                        .
                       range2 = "LabelN";
  .
  .
  .
VALUE formatM range1 = "LABEL1"
                      .
                        .
                        .
                       range2 = "LabelN";
RUN;
```

- **Format names**:
    - Are 32 characters or fewer in length.
    - Begin with a letter or underscore for formats to be assigned to numeric variables.
    - Begin with a dollar sign ($) followed by a letter or underscore for formats to be assigned to character variables.
    - Cannot end in a number.
    - Cannot conflict with the names of SAS-supplied formats.
- Note that format names do not end in a dot (.) in a VALUE statement, although they do everywhere else.

- **Format definition ranges**:

Single value or OTHER
 	
(missing values will be included in OTHER unless coded explicitly into a range)
	`proc format;`
	` value gender`
	`   1 = 'Male'`
	`   2 = 'Female'`
	`   other = 'Error';`
	` value $ gendLong`
	`   "M" = "Male"`
	`   "F" = "Female"`
	`   other = " ";`
	`run;`

List of values or an actual range	`proc format;`
	` value wdays`
	`   7,6 = 'Week End'`
	`   1-5 = 'Week Day';`
	`run;`

Ranges of values including LOW and HIGH
 	
(missing values are not included in LOW)
	`proc format;`
	` value ageCat`
	`   LOW-<65   = '<65'`
	`   65-HIGH = '>=65';`
	`run;`

- **Format definition labels**:
  - Can be up to 256 characters in length.
  - Should be enclosed in quotes (although this is not absolutely required).
- Values not in any ranges are displayed as is, unformatted.
- Except in special cases (see the MULTILABEL option), ranges should not overlap.
- The less than symbol (<) can be used in ranges to exclude either end point of the range. Some examples:
  - 1-10 (values 1-10 inclusive of each end point)
  - 1<-10 (values greater than 1 up to and including 10)
  - 1-<10 (1 up to but not including 10)
  - 1<-<10 (values greater than 1 up to but not including 10)
- User-defined formats can be used in FORMAT statements in DATA steps as well as PROC steps.
- If the data set is stored permanently, you must have the format available whenever the data set is used, since the format association is part of the permanent data set.

Figure 13: Example with User-Defined Formats

```
proc format;
       value prfmt
       0<-100     = "Low"
       100<-500   = "Medium"
       500<-<700  = "High"
       700-high   = "Very High"
       LOW-0      = "Invalid"
       OTHER      = "Missing";
       value $rfmt 'MON', 'TUE', 'WED', 'SUN' = 'No R'
                       'THR', 'FRI', 'SAT' = 'R'
                        OTHER = 'Invalid';
run;

proc print data =bios511.sales2(obs=10); run;


proc print data=bios511.sales2(obs=10);
       format price prfmt. weekday $rfmt.;
run;

proc freq data=bios511.sales2;
       tables price weekday / missing;
       format price prfmt. weekday $rfmt.;
run;
```

Unformatted

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY	SEX
1		EVER	.	.	TUE	3	MALE
2	SHOES	AGILE	95.00	40.49	WED	4	FEMALE
3	SHOES	CLEVER	65.00	33.44	WED	4	MALE
4	SHOES	CLEVER	65.00	33.44	WED	4	MALE
5	FURS	BURLEY	599.95	180.01	THR	5	MALE
6	SHOES	AGILE	.	28.07	THR	5	FEMALE

Formatted

Obs	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY	SEX
1		EVER	Missing	.	No R	3	MALE
2	SHOES	AGILE	Low	40.49	No R	4	FEMALE
3	SHOES	CLEVER	Low	33.44	No R	4	MALE
4	SHOES	CLEVER	Low	33.44	No R	4	MALE
5	FURS	BURLEY	High	180.01	R	5	MALE
6	SHOES	AGILE	Missing	28.07	R	5	FEMALE

FREQ Analysis

PRICE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Low	41	82.00	41	82.00
Medium	4	8.00	45	90.00
High	2	4.00	47	94.00
Very High	3	6.00	50	100.00

WEEKDAY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
R	25	50.00	25	50.00
No R	25	50.00	50	100.00

Creating Formatted Character Variables with the PUT Function

- One can create character variables using numeric variables and a SAS format using the PUT function.
- Syntax: PUT(*source,format*) ;
- The result of the PUT function is always a character string, but either numeric or character variables can be used as the source.
- The format must be the same type as the source.
- If the source is numeric, the resulting string is right aligned.
- If the source is character, the resulting string is left aligned.

Figure 14: Example with PUT Function

```
proc format;
       value prfmt
      0<-100    = " 1: Low"
      100<-500  = " 2: Medium"
      500<-<700 = " 3: High"
      700-high  = " 4: Very High"
      LOW-0     = "98: Invalid"
      OTHER     = "99: Missing";
       value $rfmt    'MON', 'TUE', 'WED', 'SUN' = 'No R'
                          'THR', 'FRI', 'SAT' = 'R'
                           OTHER = 'Invalid';
run;

data sales;
 set bios511.sales2;

 price_c   = put(price,prfmt.);
 weekday_c = put(weekday,rfmt.);

 keep price price_c weekday weekday_c;
run;

proc sort data = sales out = unique_values_price   nodupkey; by price_c;   run;
proc sort data = sales out = unique_values_weekday nodupkey; by weekday_c; run;

proc print data = unique_values_price   noobs; var price_c price;     run;
proc print data = unique_values_weekday noobs; var weekday_c weekday; run;
```

price_c	PRICE
1: Low	95.00
2: Medium	139.95
3: High	599.95
4: Very High	700.00
99: Missing	.

weekday_c	WEEKDAY
Invalid	
No R	TUE
R	THR

Figure 15: Three Methods for Recoding Variables w QC Check

```
** create dataset;
data voting;
input VotingRate @@;
datalines;
15.3  50.0  -9  105  97.2  .  47.8  .n  73.1  0
;
run;

proc format;
    value vote 0-25   ='1'    /* very low  */
               25<-50 ='2'    /* low       */
               50<-75 ='3'    /* medium    */
               75<-100='4'    /* high      */
               other  =' ';
run;

data votecat;
 set voting;

    /* Method 1- Use Boolean logic, first checking for valid values */
    if missing(votingrate)=0 and 0<=votingrate<=100
     then VoteCat1 = 1*(     votingrate <= 25) + 2*(25 < votingrate <= 50)
                  + 3*(50 < votingrate <= 75) + 4*(75 < votingrate      );

    /*Method 2- Use IF/THEN logic, beginning with a check for invalid values */
         if  0 <= votingrate <= 25 then VoteCat2=1;
    else if 25 <  votingrate <= 50 then VoteCat2=2;
    else if 50 <  votingrate <= 75 then VoteCat2=3;
    else if 75 <  votingrate <=100 then votecat2=4;

    /*Method 3- Use a format and the PUT function */
    VoteCat3=PUT(votingrate,vote.);

    /* QC checking using PUT Statement */
   if votingrate > .z and missing(VoteCat1) then
   put 'WAR' 'NING: Unexpected value for ' votingrate;
run;

proc sort data=votecat; by votingrate; run;

proc print data=votecat; var votingrate votecat1 votecat2 votecat3; run;

proc contents data=votecat; run;
```

Obs	VotingRate	VoteCat1	VoteCat2	VoteCat3
1	.	.	.	
2	N	.	.	
3	-9.0	.	.	
4	0.0	1	1	1
5	15.3	1	1	1
6	47.8	2	2	2
7	50.0	2	2	2
8	73.1	3	3	3
9	97.2	4	4	4
10	105.0	.	.	

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	VoteCat1	Num	8
3	VoteCat2	Num	8
4	VoteCat3	Char	1
1	VotingRate	Num	8

Conditional Execution of OUTPUT Statements

- The OUTPUT statement controls when the values in the PDV are written to the output SAS data set.
- The OUTPUT statement is optional. When an OUTPUT statement does not appear in the DATA step, SAS outputs the values of the PDV at the end of the DATA step.
- When an OUTPUT statement does appear in the DATA step, there is no automatic output at the end of the step.
- When an OUTPUT statement is executed, SAS immediately outputs the current PDV values to a SAS data set.
- An OUTPUT statement is executable, so you can output conditionally.

Figure 16: Conditional Output Execution

```
proc means data = bios511.weight_club;
 var start: end: loss;
 output out = summary1
              n= mean= / autoname;
run;

proc print data = summary1;
 var Start: End: Loss:;
run;
```

```
data weight_club;
  set bios511.weight_club;
   category = 'Starting Weight';
   catOrder = 1;
   value    = StartWeight;
     if value>. then output;

   category = 'Ending Weight';
   catOrder = 2;
   value    = EndWeight;
     if value>. then output;

   category = 'Weight Loss';
   catOrder = 3;
   value    = Loss;
     if value>. then output;
run;

proc univariate data = weight_club;
 class catOrder category;
 var value;
 output out = summary2
             n=value_n mean=value_mean;
run;

proc print data = summary2; run;
```

Obs	StartWeight_N	StartWeight_Mean	EndWeight_N	EndWeight_Mean	Loss_N	Loss_Mean
1	5	173	5	155.2	5	17.8

Obs	catOrder	category	value_n	value_mean
1	1	Starting Weight	5	173.0
2	2	Ending Weight	5	155.2
3	3	Weight Loss	5	17.8

Iterative Execution of DO Groups

- Iterative DO loops are used to repeatedly execute the statements within a DO group, changing the value of the index-variable each time.
- The number of iterations and the value of the index variable are determined by the "start", "stop" and "increment" parameters, or by the list of “values” in the following syntax:

```
DO index-variable=start TO stop BY increment;
DO index-variable=start TO stop;
DO index-variable=value1, value2, ... , valuen;
```

- Every `DO` statement must be paired with a matching statement.
- Suppose you have a DO group of the form:

```
DO index-variable=start TO stop BY increment;
                          .
                          .
END;
```

This expression is interpreted as follows:

1. When the DO statement is first encountered, the index-variable is set to "start".
2. If the index-variable is greater than "stop", then control passes to the statement following the END statement.
3. If the value of the index-variable is less than or equal to "stop", the statements in the DO group are executed.
4. At the end of the DO group, "increment" is added to the index-variable and control branches back to the test against "stop" (step 2 above).
5. This process is repeated until the index-variable is greater than the "stop" value. Control then passes to the statement following the END statement.

Figure 17: Example Using a DO Loop

```
data test;
  x=0 ;
  do i = 1 to 3 by 1;
    x = x + i;
  end;
run;

proc print data = test1 noobs; run;
```

x	i
6	4

```
data test;
  x = 0;

  /* start loop */
    i = 1;
    x = x+i;


    i = 2;
    x = x+i;

    i = 3;
    x = x+i;

    i = 4;
  /* stop loop */
run;
proc print data = test2 noobs; run;
```

x	i
6	4

- The index-variable will be included in the output data set unless it is explicitly dropped.
- "Start", "stop", and "increment" can all be arbitrarily complex expressions whose values are only evaluated once, the first time through the loop.
- The start, stop, increment, and value must be non-missing.
- Loops execute until the index exceeds the stop value. This means that the stop value must be reachable from start. You cannot have a start at 100 and a stop at 50 unless you used a negative increment.
- You can combine the various forms of the indexed DO statements, using start, stop, and optionally, increment, with one or more "value" specifications.
- DO loops can be nested within each other or a DO group.

Figure 18: Example Using a DO Loop to Compute Interest

Compute the final balance resulting from depositing a given amount (CAPITAL) for a given number of years (TERM) at a given rate of interest (RATE). Assume interest is compounded yearly.

```
data work.money;
 capital = 1000; term = 3; rate = 0.10; output;
 capital =  100; term = 5; rate = 0.15; output;
run;

proc print data = work.money noobs; run;

data work.compound;
   set work.money;
      do year=1 to term by 1;
         interest = capital * rate;
         capital = capital + interest;
      end;
      drop year interest;
run;

proc print data = work.compound noobs; run;
```

capital	term	rate
1000	3	0.10
100	5	0.15

capital	term	rate
1331.00	3	0.10
201.14	5	0.15

Figure 19: Using a DO Loop to Compute Cumulative UFO Sightings

Compute cumulative number of UFO sightings through every Monday and Thursday in the month on July using the “UFO” data set.

```
proc print data = bios511.UFO noobs label;
  label sightDate = 'Date' HowMany = 'Number of Sightings on Date';
run;

data sightings;
 set bios511.UFO;
  do date = '01Jul2004'd to '31Jul2004'd by 1;
     datec = strip(put(date,weekdate32.));
    if sightDate <= date and weekday(date) in (2,5) then output;
  end;
run;

proc univariate data = sightings noprint;
 class date datec;
 var HowMany;
 output out = cumulative_sightings sum=HowManySoFar;
run;

proc print data = cumulative_sightings noobs label;
 var datec HowManySoFar;
  label datec = 'Date' HowManySoFar = 'Cumulative Sightings to Date';
run;
```

Date	Number of Sightings on Date
05JUL2004	1
06JUL2004	1
07JUL2004	2
11JUL2004	3
16JUL2004	1
19JUL2004	5
23JUL2004	1
24JUL2004	1
29JUL2004	1

Date	Cumulative Sightings to Date
Monday, July 5, 2004	1
Thursday, July 8, 2004	4
Monday, July 12, 2004	7
Thursday, July 15, 2004	7
Monday, July 19, 2004	13
Thursday, July 22, 2004	13
Monday, July 26, 2004	15
Thursday, July 29, 2004	16

More Complex DO Loops

- The general syntax of the iterative DO statement is:

```
DO control expression 1, control expression 2, ... ;
      .
      .
      .
END;
```

where each "control expression" has the general form:

```
start TO stop BY increment
```

- If more than one control expression is included, each is executed in turn.
- The control expressions can also be abbreviated:
  - If `BY increment` is omitted, `BY 1` is assumed.
  - If `TO stop BY increment` is omitted, the loop will be executed once with "index variable=start".
- Example: Check the variable Y for the missing value codes 99, 998, and 999 and recode to a SAS missing value:

```
DO miss=99, 998, 999;
      IF y=miss THEN y=.;
END;
```

- A DO loop can also "count down." In that case "increment" is negative and "stop" must be less than "start." In such cases, the loop is repeated until the value of the index variable is less than "stop".

Figure 20: Parsing Addresses

```
data Address_list;
 infile datalines dsd dlm = "," missover;

 length firstName lastname $25 address $100 ;
 input firstName lastname address ;
 datalines;
  Jim,    Exponential,   "010 Survival Way, Censortown, NC 28731"
  Sally, Normal,         "812 Bell Court, Asymptopia, NY 71939"

  Buzz,   Poisson,       "123 Count Street, Discreteville"
 ;
run;
proc print data = Address_list noobs; run;

data parsed;
 set Address_list;

  if address > '' then count_word = count(address,',')+1;
   else count_word = 0;

  do wordNumber = 1 to count_word;
      word = scan(address,wordNumber,',');
      output;
  end;
run;
proc print data = parsed noobs; run;
```

firstName	lastname	address
Jim	Exponential	010 Survival Way, Censortown, NC 28731
Sally	Normal	812 Bell Court, Asymptopia, NY 71939
Buzz	Poisson	123 Count Street, Discreteville

firstName	lastname	address	count_word	wordNumber	word
Jim	Exponential	010 Survival Way, Censortown, NC 28731	3	1	010 Survival Way
Jim	Exponential	010 Survival Way, Censortown, NC 28731	3	2	Censortown
Jim	Exponential	010 Survival Way, Censortown, NC 28731	3	3	NC 28731
Sally	Normal	812 Bell Court, Asymptopia, NY 71939	3	1	812 Bell Court
Sally	Normal	812 Bell Court, Asymptopia, NY 71939	3	2	Asymptopia
Sally	Normal	812 Bell Court, Asymptopia, NY 71939	3	3	NY 71939
Buzz	Poisson	123 Count Street, Discreteville	2	1	123 Count Street
Buzz	Poisson	123 Count Street, Discreteville	2	2	Discreteville

Figure 21: Nested DO Loops

```
data one;
      length ageCategory $4 sex $6;
      do ageCategory = '<65','>=65';
        do sex = 'Female','Male';
            output;
        end;
      end;
run;
proc print data = one; run;
```

Obs	ageCategory	sex
1	<65	Female
2	<65	Male
3	>=65	Female
4	>=65	Male

Figure 22: Check Your Understanding of Nested DO Groups

```
data one;
      set bios511.class;

      n3 = 0;

      if sex='F' then do;
          n1 = 2;
          do n2 = 1 to 3;
              n3 = n3 + 2;
          end;
      end;

      else do;
          n1 = 4;
      end;

run;
proc print data = one; run;
```

Obs	NAME	SEX	AGE	HT	WT	n3	n1	n2
1	CHRISTIANSEN	M	37	71	195	0	4	.
2	HOSKING J	M	31	70	160	0	4	.
3	HELMS R	M	41	74	195	0	4	.
4	PIGGY M	F	.	48	.	6	2	4
5	FROG K	M	3	12	1	0	4	.
6	GONZO		14	25	45	0	4	.

Other Forms of Iterative DO Groups - DO WHILE and DO UNTIL Loops

- Two additional forms of DO loop available are the DO WHILE and DO UNTIL loops.
- These are used in cases in which you want to execute a loop as long as some logical expression is true (WHILE loop) or as long as some logical expression is false (UNTIL loop).
- Syntax:

```
DO WHILE (Expression);
    ... Executable statements ...
END;
```

```
DO UNTIL (Expression);
    ... Executable statements ...
END;
```

- In a DO WHILE loop, the expression is evaluated at the top of the loop, before the statements in the DO group are executed. If the expression is true, the DO group is executed.
- In a DO UNTIL loop, the expression is evaluated at the bottom of the loop, after the statements in the DO group are executed. If the expression is true, the DO group is not executed again. The DO group is always executed at least once.

Figure 23: Example with DO WHILE/UNTIL Loops

Count the number of years needed to double an initial amount (CAPITAL) at a given rate of interest (RATE), compounding yearly.

```
data work.compound;
 capital = 1000; rate = 0.1;  output;
 capital =  100; rate = 0.01; output;
run;

proc print data = compound; run;
```

Obs	capital	rate
1	1000	0.10
2	100	0.01

```
data double_while;
   set compound;
   total = capital; term = 0;
   do while(total < (capital*2));
      total = total + (total*rate);
      term = term + 1;
   end;
run;

proc print data = double_while; run;
```

Obs	capital	rate	total	term
1	1000	0.10	2143.59	8
2	100	0.01	200.68	70

```
data double_until;
   set compound;
   total = capital; term = 0;
   do until(total >= (capital*2));
      total = total + (total*rate);
      term = term + 1;
   end;
run;

proc print data = double_until; run;
```

Obs	capital	rate	total	term
1	1000	0.10	2143.59	8
2	100	0.01	200.68	70

Figure 24: Example with DO WHILE/UNTIL Loops

Count the number of years needed to double an initial amount (CAPITAL) at a given rate of interest (RATE), compounding yearly.

"Address_list" Data Set

firstName	lastname	address
Jim	Exponential	010 Survival Way, Censortown, NC 28731
Sally	Normal	812 Bell Court, Asymptopia, NY 71939
Buzz	Poisson	123 Count Street, Discreteville

```
data parsed_while;
 set Address_list;
 drop address;

 length word $20.;
 word        = '***';
 word_index = 0;
 do while(strip(word)>'');
     word_index = word_index + 1;
     word = scan(address,word_index,',');
       if word > '' then output;
 end;
run;
proc print data = parsed_while noobs;
run;
```

firstName	lastname	word	word_index
Jim	Exponential	010 Survival Way	1
Jim	Exponential	Censortown	2
Jim	Exponential	NC 28731	3
Sally	Normal	812 Bell Court	1
Sally	Normal	Asymptopia	2
Sally	Normal	NY 71939	3
Buzz	Poisson	123 Count Street	1
Buzz	Poisson	Discreteville	2

```
data parsed_until;
 set Address_list;
 drop address;

 length word $20.;
 word_index = 0;
 do until(strip(word)='');
     word_index = word_index + 1;
     word = scan(address,word_index,',');
       if word > '' then output;
 end;
run;
proc print data = parsed_until noobs;
run;
```

firstName	lastname	word	word_index
Jim	Exponential	010 Survival Way	1
Jim	Exponential	Censortown	2
Jim	Exponential	NC 28731	3
Sally	Normal	812 Bell Court	1
Sally	Normal	Asymptopia	2
Sally	Normal	NY 71939	3
Buzz	Poisson	123 Count Street	1
Buzz	Poisson	Discreteville	2

Arrays

- Arrays are useful when running repetitive code like the following:

```
ratio1 = verbal1/math1 ;
ratio2 = verbal2/math2 ;
ratio3 = verbal3/math3;

if date1=98 or date1=99 then date1=.;
if date2=98 or date2=99 then date2=.;
if date3=98 or date3=99 then date3=.;
```

- An array defines a group of variables given a collective name.
- Arrays are a convenient way of temporarily identifying a group of variables by assigning an alias to them.
- Calculations in the DATA step can operate on arrays just like variables.
- ARRAY statements are usually used in conjunction with DO loops so that you can execute one or more statements for each of a group of related variables.
- Syntax: `ARRAY name { subscript } <$> <length>  <elements> <(values)> ;`
  - `name` is the name of the array; it cannot be a variable or existing array.
  - `{ subscript }` describes the number and arrangement of elements in the array by using an asterisk, a number, or a range of numbers.
    - Brackets [ ] or parentheses can also be used.
    - Single-Dimensional: `array simple{3} red green yellow;`
    - Multi-Dimensional: `array x{5,3} score1-score15;`
  - The optional `$` indicates that the elements of the array are character variables. The `$` may be omitted if the variables have already been defined as character.
  - The `length` indicates the length of any variables that have not yet been assigned a length.
  - The `elements` are the names of the variables in the array. Any combination of variable lists and variable names are permitted. All elements in the array must be of the same data type.
  - The optional `values` indicate initial values for array elements. These values are separated by a comma and/or one or more blanks. Starting values do not replace variable values already known to SAS.

Figure 25: Referencing Array Elements

```
proc print data = bios511.scores; run;
```

Obs	ID	SCORE1	SCORE2	SCORE3
1	201	70	78	103
2	202	80	88	100
3	203	.	98	78
4	204	90	83	93
5	205	77	97	101

```
data _null_;
 set bios511.scores;

  array score[3] score1-score3;

  i = 1;
  x = score1;
  y = score[1];
  z = score[i];
    put ID= x= y= z=;

  i = 2;
  x = score2;
  y = score[2];
  z = score[i];
    put ID= x= y= z=;

  i = 3;
  x = score3;
  y = score[3];
  z = score[i];
   put ID= x= y= z= / /;

run;
```

```
ID=201 x=70 y=70 z=70
ID=201 x=78 y=78 z=78
ID=201 x=103 y=103 z=103


ID=202 x=80 y=80 z=80
ID=202 x=88 y=88 z=88
ID=202 x=100 y=100 z=100


ID=203 x=. y=. z=.
ID=203 x=98 y=98 z=98
ID=203 x=78 y=78 z=78


ID=204 x=90 y=90 z=90
ID=204 x=83 y=83 z=83
ID=204 x=93 y=93 z=93


ID=205 x=77 y=77 z=77
ID=205 x=97 y=97 z=97
ID=205 x=101 y=101 z=101
NOTE: There were 5 observations
      BIOS511.SCORES.
NOTE: DATA statement used (Total
      real time           0.05 s
      cpu time            0.01 s
```

- The following are examples of valid ARRAY statements:

ARRAY Statement	Note
`ARRAY test{3} test1-test3 ;`	If test1-test3 do not exist, they will be numeric and initialized to missing (.). If test1-test3 do exist, this statement is valid regardless of their type (assuming they have the same type).
`ARRAY test{3} ;`	SAS will create three numeric variables and name them test1-test3.
`ARRAY test[3] $ ;`	SAS will create three character variables, name them test1-test3, and set their respective lengths to 8.
`ARRAY day{3} $2 d1-d3 ('S','M','T') ;`	This will create 4 new variables and give d1 the initial value “S”, etc.
`ARRAY x[*] _NUMERIC_ ;`	This will put all numeric variables currently in the PDV in the array. The keyword _CHARACTER_ can also be used. The asterisk can be used when you do not want to hard code the number of variables in the array (e.g., data driven).
`ARRAY y{*} t1-t3 s4-s6 ;`	The elements of an array need not have a common naming convention.
`ARRAY scores[5,3] score1-score15 ;`	This defines a 2-d array with 5 rows and 3 columns. The variables score1-score3 will correspond to scores[1,1]-scores[1,3].

Figure 26: Normalizing Raw Test Scores

```
data scores ;
      set bios511.scores;

      array score[3] score1-score3;
      array newScore[3] ;

      do i = 1 to 3;
            newScore[i] = score[i] / 100;
      end;

      drop i;
run;
proc print data = scores; run;
```

Obs	ID	SCORE1	SCORE2	SCORE3	newScore1	newScore2	newScore3
1	201	70	78	103	0.70	0.78	1.03
2	202	80	88	100	0.80	0.88	1.00
3	203	.	98	78	.	0.98	0.78
4	204	90	83	93	0.90	0.83	0.93
5	205	77	97	101	0.77	0.97	1.01

Figure 27: Identifying a best Test Score

```
data scores ;
  set bios511.scores;

      array score[3] score1-score3;


      bestScore = .;
      bestTest  = .;

      do i = 1 to 3;
            if bestScore < score[i] then do;
         bestScore = score[i];
             bestTest  = i;
            end;
      end;

      drop i;
run;
proc print data = scores; run;
```

Obs	ID	SCORE1	SCORE2	SCORE3	bestScore	bestTest
1	201	70	78	103	103	3
2	202	80	88	100	100	3
3	203	.	98	78	98	2
4	204	90	83	93	93	3
5	205	77	97	101	101	3

Figure 28: Computing Test Grades

```
proc format;
 value grd
  LOW-<70,OTHER = 'F'
  70-<80        = 'L'
  80-<90        = 'P'
  90-HIGH       = 'H';
run;


data scores ;
  set bios511.scores;

      array score[3] score1-score3;
      array grd[3] $1;

      do i = 1 to 3;
       grd[i] = put(score[i],grd.);
      end;

      drop i;
run;
proc print data = scores; run;
```

Obs	ID	SCORE1	SCORE2	SCORE3	grd1	grd2	grd3
1	201	70	78	103	L	L	H
2	202	80	88	100	P	P	H
3	203	.	98	78	F	H	L
4	204	90	83	93	H	P	H
5	205	77	97	101	L	H	H

Figure 29: Transposing Data (Wide to Long)

```
data one;
 set bios511.scores;

 drop score:;
 rename ns = score;
 array s{*} score:;
 do i = 1 to dim(s);
   ns = s{i};
   output;
 end;
run;
```

Obs	ID	i	score
1	201	1	70
2	201	2	78
3	201	3	103
4	202	1	80
5	202	2	88
6	202	3	100
7	203	1	.
8	203	2	98
9	203	3	78
10	204	1	90
11	204	2	83
12	204	3	93
13	205	1	77
14	205	2	97
15	205	3	101

```
data two;
 set bios511.scores;
 i=1; score=score1; output;
 i=2; score=score2; output;
 i=3; score=score3; output;
 keep id i score;
run;
```

Obs	ID	i	score
1	201	1	70
2	201	2	78
3	201	3	103
4	202	1	80
5	202	2	88
6	202	3	100
7	203	1	.
8	203	2	98
9	203	3	78
10	204	1	90
11	204	2	83
12	204	3	93
13	205	1	77
14	205	2	97
15	205	3	101

```
proc transpose
 data = bios511.scores
  out = three;
 by id;
 var score:;
run;
```

Obs	ID	_NAME_	COL1
1	201	SCORE1	70
2	201	SCORE2	78
3	201	SCORE3	103
4	202	SCORE1	80
5	202	SCORE2	88
6	202	SCORE3	100
7	203	SCORE1	.
8	203	SCORE2	98
9	203	SCORE3	78
10	204	SCORE1	90
11	204	SCORE2	83
12	204	SCORE3	93
13	205	SCORE1	77
14	205	SCORE2	97
15	205	SCORE3	101

- You can use the DIM function to return the number of elements in a dimension of an array.

Figure 30: Transposing Data (Long to Wide)

```
data four;
 set two;

 retain score1-score3;

 drop i score;

 if i=1 then score1=score;
 else if i=2 then score2=score;
 else if i=3 then score3=score;

 if i=3 then output;
run;
```

```
data five;
 set two;

 retain score1-score3;
 array s[3] score1-score3;

 drop i score;

 s[i] = score;
 if i=3 then output;
run;
```

```
proc transpose data   = two
               out    = six
               prefix = score;
 by id;
 id i;
 var score;
run;
```

Obs	ID	score1	score2	score3
1	201	70	78	103
2	202	80	88	100
3	203	.	98	78
4	204	90	83	93
5	205	77	97	101

Obs	ID	_NAME_	score1	score2	score3
1	201	score	70	78	103
2	202	score	80	88	100
3	203	score	.	98	78
4	204	score	90	83	93
5	205	score	77	97	101

Figure 31: Shorthand for Summary Statistical Functions

```
data toy_example;

 array x[5] a b c d e (1 . 23 -14 9);

 y1 = sum(of x[*]); y2 = sum(a,b,c,d,e);
 z1 = min(of x[*]); z2 = min(a,b,c,d,e);

run;
proc print data = toy_example; run;
```

Obs	a	b	c	d	e	y1	y2	z1	z2
1	1	.	23	-14	9	19	19	-14	-14

Figure 32: Using Arrays to Define Temporary Constants

```
data one;

   array test{6} _temporary_ (90 80 70 70 95 60);
   array grade{6} $;

   do j = 1 to dim(test);
      if test{j} >= 95 then grade{j} = 'a';
      else if test{j} >= 85 then grade{j} = 'b';
      else if test{j} >= 75 then grade{j} = 'c';
      else if test{j} >= 70 then grade{j} = 'd';
      else grade{j} = 'f';
   end;

drop j;
run;

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

grade1	grade2	grade3	grade4	grade5	grade6
b	c	d	d	a	f

Transformations Involving Missing Values

- Missing values occur in most data, and it is important to understand what effect these missing values have on transformations of variables.

- Missing values for numeric variables are:

  - Presented in programming statements by: `._, ., .A-.Z`

  - Checked for missing by:

```
IF  x<=.z THEN DELETE;
IF MISSING(x) THEN DELETE;
```

- Note that a numeric missing value is the numeric value ., not a character string for "."

- Missing value for character variables are:

  - Represented by a blank field (‘’)

  - Checked for missing by:

```
IF g=' ' THEN DELETE;
IF MISSING(g) THEN DELETE;
```

- Variables can be assigned missing values.

```
IF wt GT 500 THEN wt = .;
IF wt GT 500 THEN wt=.B;
IF state NE 'NC' THEN state = ' ';
```

- Missing values propagate through arithmetic expressions.

- Illegal arguments sent to functions return missing values.

Missing Values Example 1

```
DATA work.b;
	SET work.a;
	z = x+y;
	s = SQRT(x);
RUN;
```

Dataset A

x	y
4	2
.	7
-3	2

Dataset B

x	y	z	s
4	2	6	2
.	7	.	.
-3	2	-1	.

- Variable x is missing in the third observation, therefore:
  - Variable z is missing, since applying the addition operator to a missing value results in a missing value.
  - Variable s is missing, since the square root of a missing value is missing.

- Variable x is negative in the fourth observation, and a square root of a negative number is not defined, therefore variable s is missing.

Missing Values Example 2

- Missing values are treated as less than minus infinity in comparison expressions.

- Special missing values compare in the sort sequence.

```
DATA work.d1;
	SET work.c;

	LENGTH $ t1 5;
	IF x < y THEN t1='TRUE';
	ELSE t1='FALSE';

RUN;
```

Dataset C

x	y
0	4
-12	.
.	9
A	B

Dataset D1

x	y	t1
0	4	TRUE
-12	.	FALSE
.	9	TRUE
A	B	TRUE

- Note that the "A" and "B" values of the variable x and y are <u>not</u> character strings. They can't be character strings, since x and y are numeric variables! Instead the "A" and "B" are the missing values .A and .B. Notice how SAS prints them without the dot in front.

Missing Values Example 3

- Missing values are false in logical operations.

```
DATA work.d2;
	SET work.c;

	LENGTH t2 $5;
	IF x THEN t2='TRUE ';
	ELSE t2='FALSE';

RUN;
```

Dataset C

x	y
0	4
-12	.
.	9
A	B

Dataset D2

x	y	t2
0	4	FALSE
-12	.	TRUE
.	9	FALSE
A	B	FALSE

Missing Values Example 4

- Special missing values compare in the sort sequence (an additional example).

```
DATA work.d3;
	SET work.c;

	LENGTH t3 $5;
	IF y < .b THEN t3='TRUE';
	ELSE t3='FALSE';

RUN;
```

Dataset C

x	y
0	4
-12	.
.	9
A	B

Dataset D3

x	y	t3
0	4	FALSE
-12	.	TRUE
.	9	FALSE
A	B	FALSE

Missing Values Example 5

- You can run simple code to see whether a variable has any special missing values.

```
*Create some data;
*More on the INPUT statement later in the course;
DATA one;
INPUT x @@;
DATALINES;
1 . .z .n 3 4 .z
;

PROC FREQ DATA=one;
	WHERE MISSING(x);
	TABLES x / MISSING;
RUN;
```

The FREQ Procedure

x	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	1	25.00	1	25.00
N	1	25.00	2	50.00
Z	2	50.00	4	100.00

Missing Values Example 6

- Functions that compute sample statistics use only non-missing values of the arguments.

```
DATA work.f;
	SET work.e;

	tot = a + b + c;
	ave = tot/3;

	s = SUM(a,b,c);
	m = MEAN(a,b,c);

RUN;
```

Dataset E

a	b	c
3	2	7
.	4	9

Dataset F

a	b	c	tot	ave	s	m
3	2	7	12	4	12	4.0
.	4	9	.	.	13	6.5

Missing Values Example 7

- The SUM function can be used to prevent cumulative totals of variables involving missing values from becoming missing.

```
DATA accumulate;
        SET bios511.class;

        RETAIN cumht cumwt cumage;

        cumht  = SUM(cumht, ht);
        cumwt  = SUM(cumwt, wt);
        cumage = SUM(cumage, age);

RUN;
```

Dataset ACCUMULATE

NAME	SEX	AGE	HT	WT	cumht	cumwt	cumage
CHRISTIANSEN	M	37	71	195	71	195	37
HOSKING J	M	31	70	160	141	355	68
HELMS R	M	41	74	195	215	550	109
PIGGY M	F	.	48	.	263	550	109
FROG K	M	3	12	1	275	551	112
GONZO		14	25	45	300	596	126

The SUM Statement

A sum statement can be used to sum expressions over observations. It implies a RETAIN and only sums nonmissing values.

Syntax:

```
Sum_variable + Expression;
```

Example:

```
DATA accumulate2;
	SET classlib.class;

	cumht + ht;
	cumwt + wt;
	cumage + age;

RUN;
```

Data Set ACCUMULATE2 (just like ACCUMULATE)

NAME	SEX	AGE	HT	WT	cumht	cumwt	cumage
CHRISTIANSEN	M	37	71	195	71	195	37
HOSKING J	M	31	70	160	141	355	68
HELMS R	M	41	74	195	215	550	109
PIGGY M	F	.	48	.	263	550	109
FROG K	M	3	12	1	275	551	112
GONZO		14	25	45	300	596	126