Creating SAS Data Set

STAT2005

Chapter 9

SAS data sets

- There are two types of SAS data sets: temporary SAS data sets and permanent SAS data sets.
- Temporary SAS data set: SAS data set that could only be used within the interactive session and will be deleted when the session terminates.
- Permanent SAS data set: SAS data set that is stored in a disk drive and can be used in another SAS session after the current session terminates.
- A SAS data set is stored in a Library. Whether a SAS data set is temporary or permanent is determined by the library in which the data set is stored.

- Each library physically links up to a memory location (e.g. `C:\MyData`). `WORK` is a built-in library in SAS to store temporary SAS data sets. All data sets in `WORK` will be deleted when the session terminates.
- Data sets in other libraries are permanent. There are other built-in libraries, such as SASUSER, which we should not be using except as intended.
- Apart from the built-in libraries, we can define our own libraries. For example we can define library AB which links to `D:\` and `CD` to `C:\MyData` (a directory `MyData` in `C` drive).
- The name of user-defined library can be changed from session to session. Therefore in another session, we may link library EF to `D:\`.

- Complete SAS file name has the format `ref.filename`, where `ref` is called the first-level name or libref, and `filename` is called the second-level name.
- The first-level name identifies the library where the file is stored, and the second-level name identifies the specific file.
- For example `IN.OBSERV` is a valid data set name with `IN` being the name of the library and `OBSERV` being the filename. If the first-level name is not specified, it is assumed to be `WORK`. Therefore, the full data set name for `FITNESS` in Chapter 8 Example 2 is `WORK.FITNESS`.

- The first-level name can be changed from session to session as far as they link to the same directory.
- As the second-level name refers to the actual filename, we have to use the same second-level name in order to get the correct data set.
- Permanent SAS data set must have two-level data set name with a libref other than WORK, e.g. IN.OBSERV.
- The file name of a SAS data set has the extension sas7bdat. Therefore if IN refers to D:\, then the data set IN.OBSERV is stored in D:\ as OBSERV.sas7bdat.

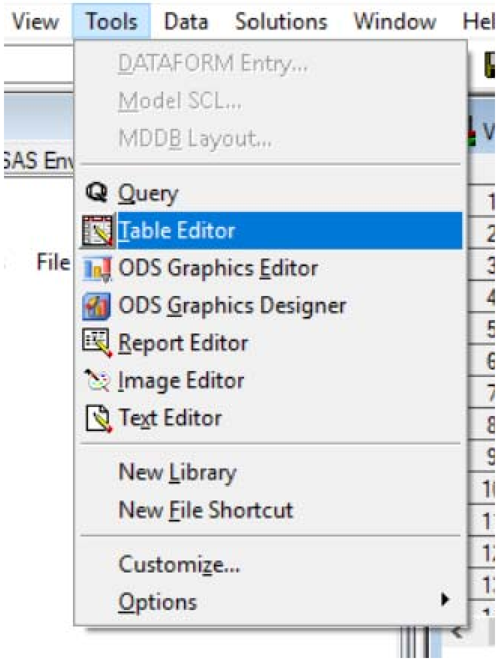
Methods for reading data

We shall discuss 3 ways to read in data and create SAS data sets:

- Entering data directly in the Viewtable window
- Using Import Wizard to read in data of other software formats
- Using DATA step

Using Viewtable window

- You can use the Viewtable window to enter your data in a tabular format.
- You can define variables and give them attributes such as name, length, and type (character or numeric).
- To open the Viewtable window, select Table Editor from the Tools menu.



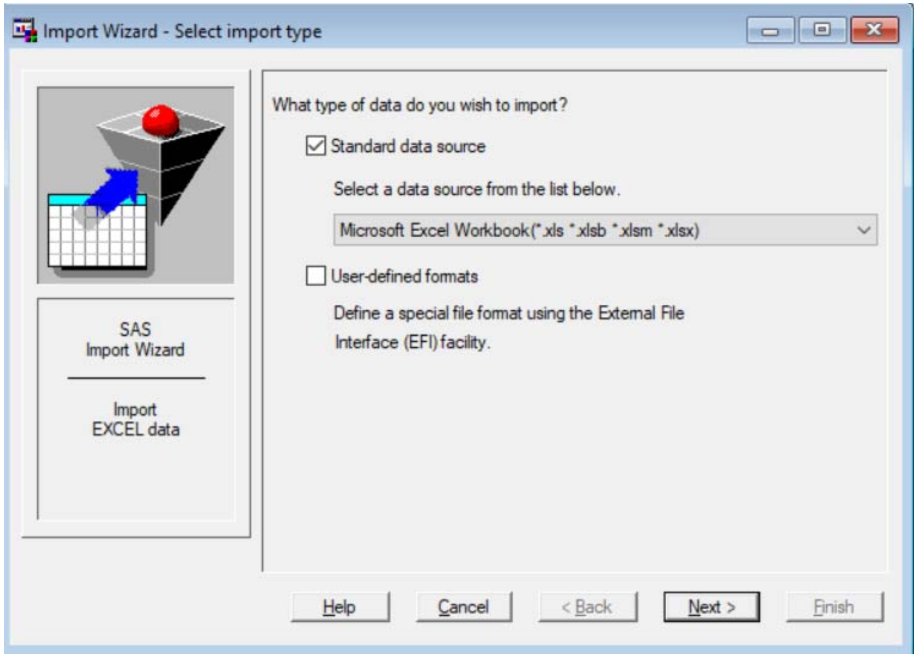
- You can open the Column Attributes window by right-clicking on the header of a column to set up column attributes.
- When you finished entering or editing your new or existing data set, you can close it. SAS will urge you to save the changes, then save the data set in a library you want.
- For example, if you saved your table in the SASUSER library and named it as HELLO, you could print it with the following program:

```
PROC PRINT DATA=SASUSER.HELLO;
RUN;
```

Using Import Wizard

- You can read a variety of data file types into SAS using the Import Wizard, which can read all types of delimited files including CSV files and Excel XLS/XLSX files.

- To start the Import Wizard, select Import Data from the File menu.



Import Wizard - Select import type

What type of data do you wish to import?

☑ Standard data source

Select a data source from the list below.

Microsoft Excel Workbook(*.xls *.xlsb *.xlsm *.xlsx)

□ User-defined formats

Define a special file format using the External File Interface (EFI) facility.

- Select the type of file you are importing by choosing from the list of standard data sources. Some common formats are Excel .xls .xlsx files, comma-separated values .csv files.
- Next, specify the location of the file that you want to import.
- For delimited files, specify the delimiter in the Delimiter box of the Delimited File Options window by clicking the Options... button .
- For CSV or tab-delimited files, the delimiter is already determined so that section of the window is grayed out.

Using DATA step

- You will have the most flexibility to read the data using the DATA step if they are in raw data files.
- By the way, Spreadsheet files are examples of data files that are NOT raw data.
- Recall the data set in Example 2 of Chapter 8, which will be referred as fitness example below:

```
DATA fitness;
  INPUT name $ weight waist pulse chins situps jumps;
  CARDS;
  Hodges    191  36  50   5  162   60
  Kerr      189  37  52   2  110   60
  Putnam    193  38  58  12  101  101
  Roberts   162  35  62  12  105   37
  Blake     189  35  46  13  155   58
RUN;
```

LIBNAME statement

- We can use a LIBNAME statement to set up a new Library. It associates a libref with the directory where the SAS files are stored. The syntax of LIBNAME is

```
LIBNAME ref 'pathname';
```

- An example of LIBNAME statement is

```
LIBNAME ABC 'D:\';
```

- Note that 'pathname' can be replaced by "pathname". We will assume the equivalence of ' ' and " " throughout the notes unless exceptional cases which are explicitly stated.
- In the fitness example, the fitness data is stored in Work library.
- However if we defined a new library ABC as above and if we replace fitness by ABC.fitness in the program, SAS will store it in D:\ drive with the filename fitness.sas7bdat.

In-stream data and CARDS, CARDS4 statements

- Let us consider a data set that comes with the SAS program like the fitness example. We call this kind of data, in-stream data.
- In-stream data must be put at the end of the DATA step. We use a CARDS or CARDS4 statement to tell SAS where the data start. The syntax of CARDS and CARDS4 statements are

```
      CARDS;
      CARDS4;
```

Both CARDS and CARDS4 statements tell SAS that the raw data begins in the next line. For CARDS statement, the data lines end by a line having a ";". Therefore, there should not be any semicolon (;) in the data lines.

- If our data lines have data value that includes ";", we have to use a CARDS4 statement instead of a CARDS statement. When CARDS4 is used, the data lines end by a line containing ;;;; in the first four columns.
- Note that another statement

```
    DATALINES;
```

can be used in place of the CARDS statement. Similarly

```
    DATALINES4;
```

can be used in place of the CARDS4 statement.

External text file and the `INFILE` statement

- If our data is already stored in an existing text file, we can, of course, copy the text file to the Editor and read the data as an in-stream data.
- However, a standard method is to instruct SAS to read the data from the text file directly. To do so, we need to use an `INFILE` statement.

```
    INFILE 'filename';
```

The `INFILE` statement identifies an external file that we want to read. The `INFILE` statement must be executed before we read the data lines from the file.

- Suppose that our text file is `C:\SASDATA\sample.txt` (`C` : drive; `SASDATA` : directory; `sample.txt` : filename).
- Then the INFILE statement should read as

```
    INFILE 'C:\SASDATA\sample.txt';
```

Raw Data Input

We have three styles of raw data input that can be handled by the DATA Step:

- List input,
- Column input,
- Formatted input.

These methods can be used with combination in practice.

Usually, raw data are arranged in one of the following two styles

- Data values come consecutively and are not aligned across lines. The list input should be used. E.g.,

```
AAAAAA BBBBB CC DDDD
AAAA BBB CCCCC DDDD
AAA BBBBB CCCC DDDDDD
```

- Data values are aligned across lines. The column input should be used. E.g.,

```
AAAAAA BBBBB CC     DDDD
AAAA   BBB   CCCCC  DDDD
AAA    BBBBB CCCC   DDDDDD
```

Reading data: List input

- The simplest way to input a small set of data is to read it using list input. Data are read in free format. Values are listed one by one, separated by one or more blanks.
- Usually we have to start a new data line for each record.

Reading in-stream data:

```
[LIBNAME ref 'pathname';]
DATA [dsname];
[LENGTH {varlist [$] length} ... ;]
INPUT {variable [$ [&]] [/]} ... [@@];
<We can put some other statements here.>
CARDS | CARDS4 ;
<Enter our data here.>
[;;;;]
RUN;
```

Reading data from external text file:

```
[LIBNAME ref 'pathname';]
DATA [dsname];
INFILE 'filename';
[LENGTH {varlist [$] length} ... ;]
INPUT {variable [$ [&]] [/]} ... [@@];
<We can put some other statements here.>
RUN;
```

Notations for syntax of SAS statements

- A vertical bar, "|", indicates that a choice among the items, separated by bar(s), must be made, e.g. a | b | c means either a or b or c.
- An ellipsis, "...", indicates that the item just described may be repeated.
- Square brackets, "[ ]", indicate optional items. If the item is a required one, we shall not type the square brackets.
- Braces "{}" is used to define an item (a group of objects) in order to eliminate ambiguity when the ellipsis or vertical bar are used.

Explaining the syntax

```
[LIBNAME ref 'pathname';]
```

- LIBNAME statement: Define library. This statement is optional. We use it when we want to define a library where we store permanent SAS data file(s) or retrieve existing permanent SAS data file(s).

```
DATA [dsname];
```

- DATA statement: The item "dsname" is the name of the data set to be created in the DATA step.
- If dsname is IN.RECORD, the SAS data set created will be a permanent one locating in the library IN with data set name RECORD.
- If dsname is RECORD, SAS will assume that it is WORK.RECORD, which is a temporary file.
- dsname is optional. If we do not specify a dsname, SAS will assign to the data set a dsname which takes the following form: WORK.DATA1, WORK.DATA2, .. .

```
    INFILE 'filename';
```

- `INFILE` statement: If we input data from a text file, `INFILE` statement tells SAS the filename of the text file.

```
    [LENGTH {varlist [$] length} ... ;]
```

- `LENGTH` statement: `LENGTH` statement is optional. It specifies the order, the variable type and amount of storage (in bytes) used by variables in a SAS data set.
    - `varlist` is a list of variables. The variables in the list are character if followed by "$"; otherwise they are numeric.
    - `length` specifies the length, in bytes, of the variables in the `varlist`. For character constant, one byte stores one character. Therefore, if the value of a character variable can have 10 characters, we need to use at least 10 bytes to store the value of this variable.
    - By default, SAS uses 8 bytes for variables.
- We can explicitly define the length of a character variable using a `LENGTH` statement. For character variables, the length can be from 1 to 32767 under all operating environments. For numeric variables, the length can be from 3 to 8.

```
INPUT {variable [$ [&]] [/]} ... [@@];
```

- INPUT statement: The INPUT statement specifies the name, the variable type, as well as the order of the variables to be read.
- The symbol "$" indicates that the preceding variable is a character variable. Variable which is not followed by a "$" is assumed to be numeric.
- The symbol "&" indicates that the character value of the preceding variable may have **one or more single embedded blank** and is to be read from the next nonblank column until the pointer reaches **two consecutive blanks** or the end of the input line whichever comes first.
- The symbol "/" asks SAS to jump to the next input line to read the remaining variable values for the same record. It will be useful if in the data file, the variables of a record is presented in two or more lines.

- To inform SAS to hold the current data line for next INPUT operation, we use @@. This is useful if there are more than one record being put in a single line. @@ must be placed at the end of the INPUT statement.

```
    CARDS  |  CARDS4  ;
```

- CARDS or CARDS4 statement: These two statements signal that the raw data begins in the next line. The data must be placed at the end of the data step.

```
    RUN;
```

- RUN statement: This statement causes the previously entered SAS statements to be executed. Although this statement is optional in some cases, it is a good habit to include a RUN statement at the end of each DATA step or PROC specification.

Example 1

Consider the following program.

```
LIBNAME DATAIN "D:\SAS";

DATA DATAIN.NEW;
INPUT X Y;

CARDS;
2   4
3   8
RUN;
```

- DATAIN is a libref to the path "D:\SAS". The SAS data set, DATAIN.NEW, is stored in D:\SAS\new.sas7bdat.
- The data set has two variables X and Y and two observations. For the first observation, X = 2 and Y = 4. For the second observation, X = 3 and Y = 8.
- As the data set is stored as a permanent file, we can, on another session, submit the following program to print the above data set.

```
LIBNAME NEWDATA 'D:\SAS';
PROC PRINT DATA=NEWDATA.NEW;
RUN;
```

Question: What will happen if the data lines are as follows?

```
1
2   4
3    8
```

The data set will have two records as shown below:

x	y
1	2
3	8

It is because

(i) SAS will move to the next data line when the data cannot be found in the current data line.

(ii) SAS moves to a new data line to read a new record.

Note that the value 4 in the second data line would not be read.

Example 2

The following program creates a permanent data set called SCORES in D:\SAS.

```
LIBNAME STUDENT 'D:\SAS';
DATA STUDENT.SCORES;
LENGTH NAME $ 14;
INPUT NAME $ & AGE SEX $ GRADE;
CARDS;
CHAN CHI SHING  15 M 70.5
CHEUNG TIN  16 M 67.2
LEUNG MAN SHAN  18 F 72.0
MA YUEN LAN  . F 80.1
WU KEUNG  17 M 75.3
RUN;
```

- In this example, we find that there are one or more single embedded blank in the values of NAME. To handle this problem, we add an "&" symbol after the variable NAME in the input statement.
- It tells SAS to end reading NAME until two consecutive blanks are encountered.

Question: For the above program, what is the order of the variables stored in the file?

Answer: The order of variables in the data set depends on the order of the variable made known to SAS. For this data set, the order should be NAME, AGE, SEX, GRADE.

Therefore, next time when we refer to this data set, AGE -- GRADE means AGE SEX GRADE.

Question: What will happen if the LENGTH statement is removed from the above program?

Answer: After removing the LENGTH statement, the data are as given below

```
CHAN CHI 15 M 70.5
CHEUNG T 16 M 67.2
LEUNG MA 18 F 72.0
MA YUEN . F 80.1
WU KEUNG 17 M 75.3
```

because by default, NAME has length 8.

Because of "&", SAS will start reading next item after the two consecutive blanks. Thus the values for AGE are read correctly.

Question: What will happen if '&' is removed from the `INPUT` statement?

Answer: We will see the following note in the Log window indicating that the program has problems in reading some values.

```
NOTE: Invalid data for AGE in line 246 8-10.
RULE: ----+----1----+----2----+----3----+----4--
--+----5----+----6----+----7----+----8----+--
246 CHAN CHI SHING 15 M 70.5
NAME=CHAN AGE=. SEX=SHING GRADE=15 _ERROR_=1
_N_=1
```

For the first record, SAS treats `CHI` as the value of the numeric variable AGE, and SEX becomes `SHING`.

Question: What will happen if the second data line is as follows?

```
CHEUNG TIN 16 M 67.2
```

Answer: As there are no consecutive blanks, SAS will treat the whole line as the value of NAME. As NAME has length 14, only the leading 14 characters are stored.

Therefore, NAME = "CHEUNG TIN 16".

SAS reads the value of AGE of the second record in the third data line, and therefore an error message is generated.

The value for SEX of the second record is MAN.

Question: What will happen if the "$" after NAME in the input statement is removed?

Answer: It has no effect as the LENGTH statement has already told SAS that NAME is a character variable.

Consider the following program

```
LIBNAME STUDENT 'D:\SAS';
DATA STUDENT.SCORES;
LENGTH NAME $ 14;
INPUT NAME $ & AGE SEX $ GRADE @@;
CARDS;
CHAN CHI SHING  15 M 70.5 CHEUNG TIN  16 M 67.2
LEUNG MAN SHAN  18 F 72.0 MA YUEN LAN  . F
80.1 WU KEUNG  17 M 75.3
RUN;
```

This program is equivalent to the original one except that an input data line may contain variable values of more than one record.

"@@" informs SAS to go on reading the current data line. Note also that the values for the fourth record are in the second and third data lines.

Question: What will happen if @@ is removed?

Answer: Without @@, SAS starts reading each new record from a new data line.

Therefore, `SCORES` has only three records.

The information for `CHEUNG  TIN` and `MA  YUEN LAN` will not be read.

In reading the third record, SAS treats

```
80.1 WU KEUNG
```

as `NAME`.

The following program creates the same data set as before.

```
DATA;
LENGTH NAME $ 14;
INPUT NAME $ & / AGE SEX $ GRADE;
CARDS;
CHAN CHI SHING
15 M 70.5
CHEUNG TIN
16 M 67.2
LEUNG MAN SHAN
18 F 72.0
MA YUEN LAN
. F 80.1
WU KEUNG
17 M 75.3
RUN;
```

The difference with the previous versions are

(1) `dsname` is not specified, so that SAS will assign one for us, say `WORK.DATA1`.

(2) Two input data lines are used to store the variable values of one observation, and hence we use the symbol `"/"` in the input statement to tell SAS to read the values for AGE, SEX and GRADE in the next data line.

The data can also be correctly read by replacing the `INPUT` statement with the following `INPUT` statements:

(i) `INPUT NAME $ &; INPUT AGE SEX $ GRADE;`

(ii) `INPUT #2 AGE SEX $ GRADE #1 NAME $ &;`

#n ask SAS to go to the n-th data line of the current record

Example 3

Consider a data file `ToadJump.dat` from a contest.

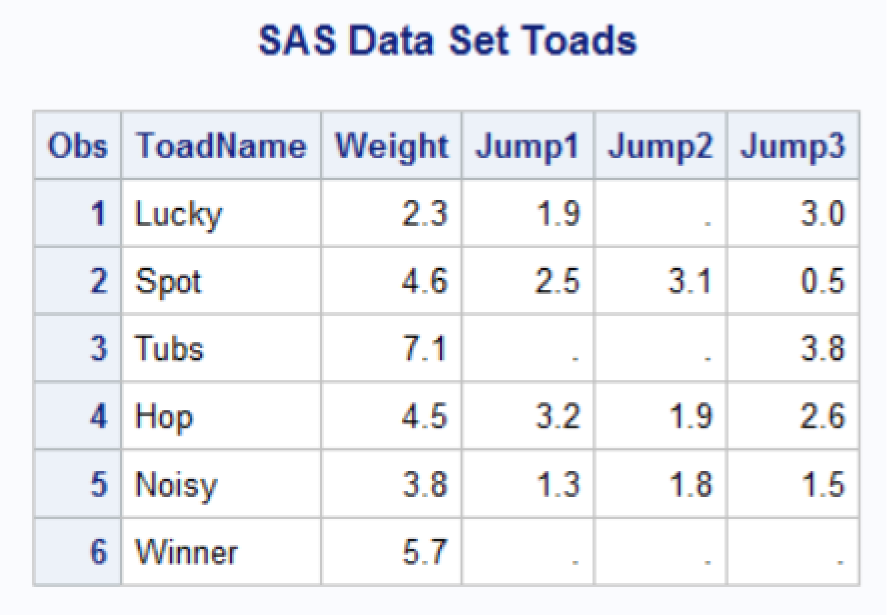
```
Lucky 2.3 1.9 . 3.0
Spot 4.6 2.5 3.1 .5
Tubs 7.1 . . 3.8
Hop 4.5 3.2 1.9 2.6
Noisy 3.8 1.3 1.8
1.5
Winner 5.7 . . .
```

For each contestant you have the toad’s name, weight, and the jump distance from three separate attempts. If the toad is disqualified for any jump, then a period is used to indicate missing data.

Note that the data for Noisy have spilled over to the next data line. This is not a problem since, by default SAS will go to the next data line to read more data if there are more variables in the INPUT statement than there are values in the data line.

Here is the SAS program that will read the data:

```
DATA toads;
INFILE 'D:\SAS\ToadJump.dat';
INPUT ToadName $ Weight Jump1 Jump2 Jump3;
RUN;
* Print the data to make sure the file was read
correctly;
PROC PRINT DATA = toads;
TITLE 'SAS Data Set Toads';
RUN;
```



SAS Data Set Toads

Obs	ToadName	Weight	Jump1	Jump2	Jump3
1	Lucky	2.3	1.9	.	3.0
2	Spot	4.6	2.5	3.1	0.5
3	Tubs	7.1	.	.	3.8
4	Hop	4.5	3.2	1.9	2.6
5	Noisy	3.8	1.3	1.8	1.5
6	Winner	5.7	.	.	.

Because SAS had to go to a second data line to get the data for Noisy’s final jump, the following note appears in the SAS log:

```
NOTE: SAS went to a new line when INPUT
statement reached past the end of a line.
```

Example 4

The following program demonstrates the use of CARDS4 statement.

```
DATA TEST;
LENGTH REGION $ 17;
INPUT NAME $ REGION $ &;
CARDS4;
TOM WONG TAI SIN; KLN
MARY SHATIN; NT
;;;;
RUN;
```

Note that if `;;;;` does not start at the first column, SAS will treat `;;;;` as part of the data.

Reading data: column input

- Column input means that the data values are arranged in a neat way that the values of each variable always appear in specific column(s) in the input lines.
- Reading data with column input has the advantages that

i. the data appear in a more uniform format, and thus it is easier for the user to check for typing error, and

ii. user can tell SAS to skip some columns, or to read certain columns more than one time.

```
INPUT {variable [$] start_column [-
end_column] [/]}... ;
```

- INPUT statement: The meanings of "$" and "/" are identical to those in list input. This statement tells SAS where to start reading a variable by specifying the starting column (start_column) and the ending column (end_column) (end_column ≥ start_column).
- The ending column is optional if the variable is one column wide.
- As the number of columns for each character variable is explicitly specified in the INPUT statement, SAS will assign the correct length to each character variable. Therefore, it is usually not necessary to include a LENGTH statement.

The following statements are invalid. (A, B, and C are variable names.)

```
INPUT A 6 – 2 B 10;
```

Invalid: Cannot use "6 – 2" as the second integer is smaller than the first.

```
INPUT A 3 @@ B 5-10;
```

Invalid: "@@" should be entered at the end.

```
INPUT A 2 B 3 C4 - 5;
```

Invalid: C4 is treated as a variable, and thus SAS does not know what "-" means; we should insert a blank between "C" and "4".

```
INPUT A $ 7-9, B 1
```

Invalid: no comma, and missing ";".

Example 5

The following program creates the same data set SCORES with column input being used. Embedded blanks can be read in column input. As SEX occupies only one column, we do not need to specify `end_column` for this variable.

```
LIBNAME STUDENT 'D:\SAS';
DATA STUDENT.SCORES;
INPUT NAME $ 1-15 AGE 17-18 SEX $ 20 GRADE 22-25;
CARDS;
CHAN CHI SHING  15 M 70.5
CHEUNG TIN      16 M 67.2
LEUNG MAN SHAN  18 F 72.0
MA YUEN LAN      . F 80.1
WU KEUNG        17 M 75.3
RUN;
```

Question: What will happen if the " **.** " in the fourth record is replaced with blank?

Answer: It has no effect, as SAS will assign a missing numeric value to AGE if columns 17 and 18 are blanks.

Example 6

```
LIBNAME STUDENT 'D:\SCHOOL';
DATA STUDENT.SCORES;
INPUT AGE 17-18 NAME $ 1-15 SEX $ 19 GRADE 21-24;
CARDS;
CHAN CHI SHING  15M 70.5
CHEUNG TIN     *16M 67.2
LEUNG MAN SHAN  18F 72.0
MA YUEN LAN      .F 80.1
WU KEUNG        17M 75.3
RUN;
```

- This programme creates the same data set STUDENT except that the order of variables has changed.
- For column input, the order of the variables to be read need not be identical to the order of their variable values appeared in the input data lines.
- In this example, we read AGE first, then NAME, then SEX and finally GRADE.
- Column 16 is not read, therefore the "*" in the second data line is not read.
- Note that there is no space between the variable values of AGE and SEX.

Example 7

We must be careful when we use @@. The following example reads the data line "1234" infinitely many times and the program would not stop.

```
DATA;
INPUT X 1-4 @@;
CARDS;
1234
2345
RUN;
```

Reading data: formatted input

- Occasionally we may have to read numeric values such as $12,000.00 and 10:30:00 (stands for 10:30 a.m.). To read this kind of values, we have to use format input.

```
INPUT {[@n] variable informat [/]} ... ;
```

- The value n after "@" must be a positive integer. @n asks SAS to jump to column n. Informat stands for input format.

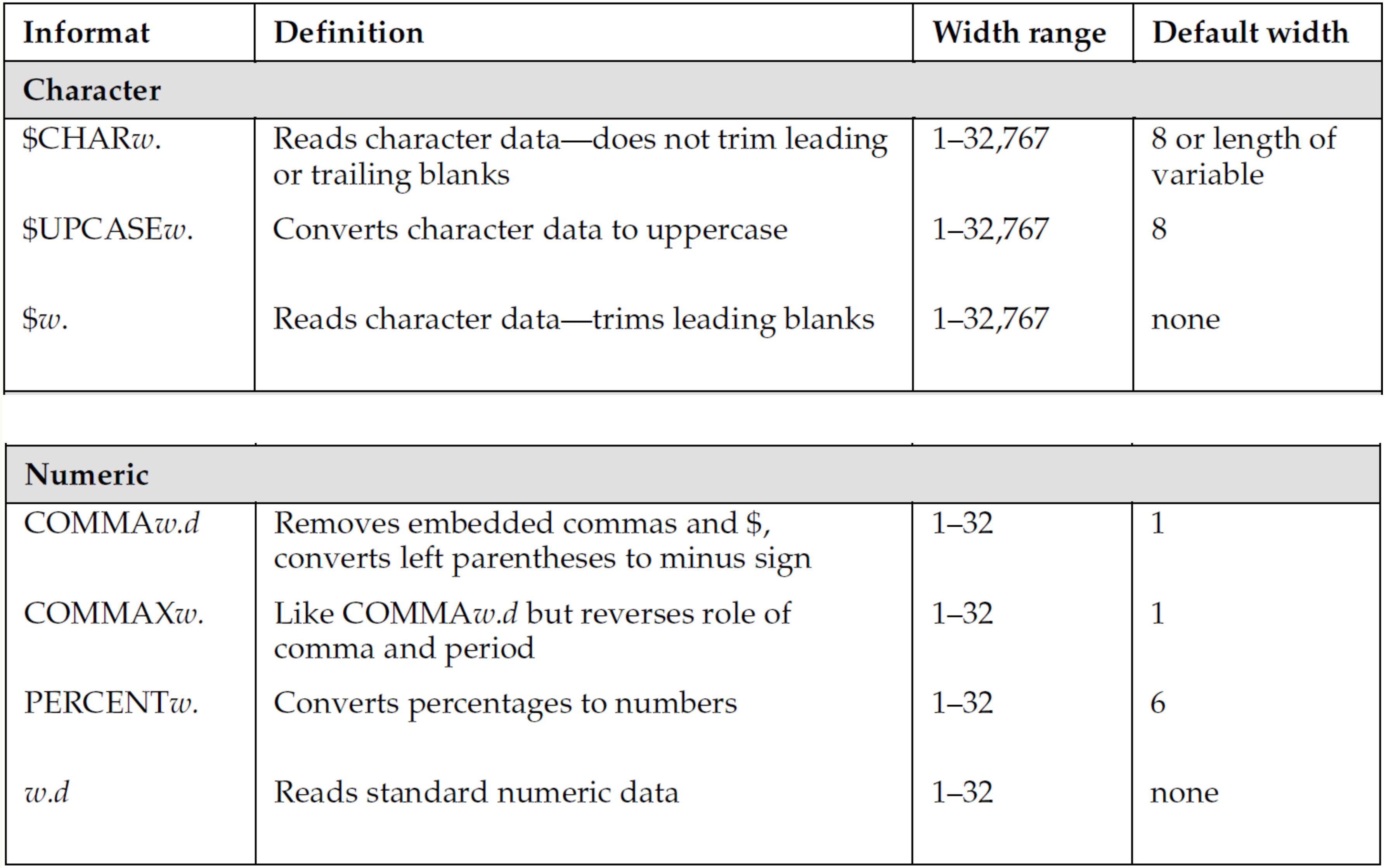
Informat

- Informats are useful when we have non-standard data.
- Dates are perhaps the most common non-standard data. Using date informats, SAS will convert conventional forms of dates like `10-31-2013` or `31OCT13` into a number, the number of days since January 1, 1960.
- There are three general types of informats: character, numeric, and date.
  - Character: `$informatw.`
  - Numeric: `informatw.d`
  - Date: `informatw.`

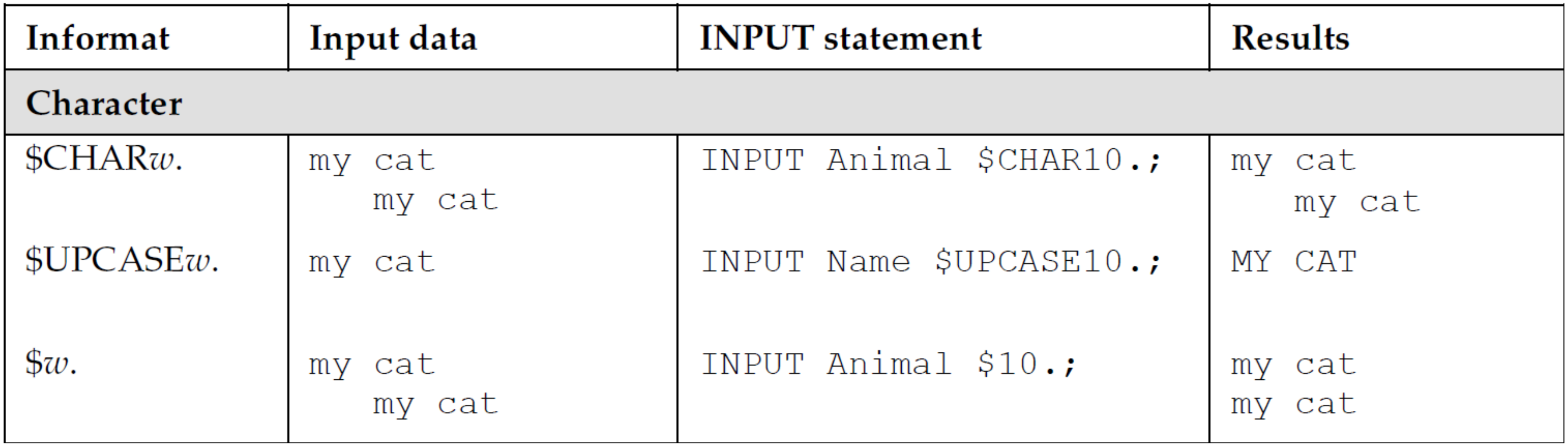
- The $ indicates character informats, `informat` is the name of the informat, `w` is the total width, and `d` is the number of decimal places (numeric informats only). Note that `d` < `w`.
- The period `"."` is a very important part of the informat name. Without a period, SAS may try to interpret the `informat` as a variable name.
- The following `INPUT` statement is an example of formatted input

```
INPUT Name $10. Age 3. Height 5.1
BirthDate MMDDYY10.;
```

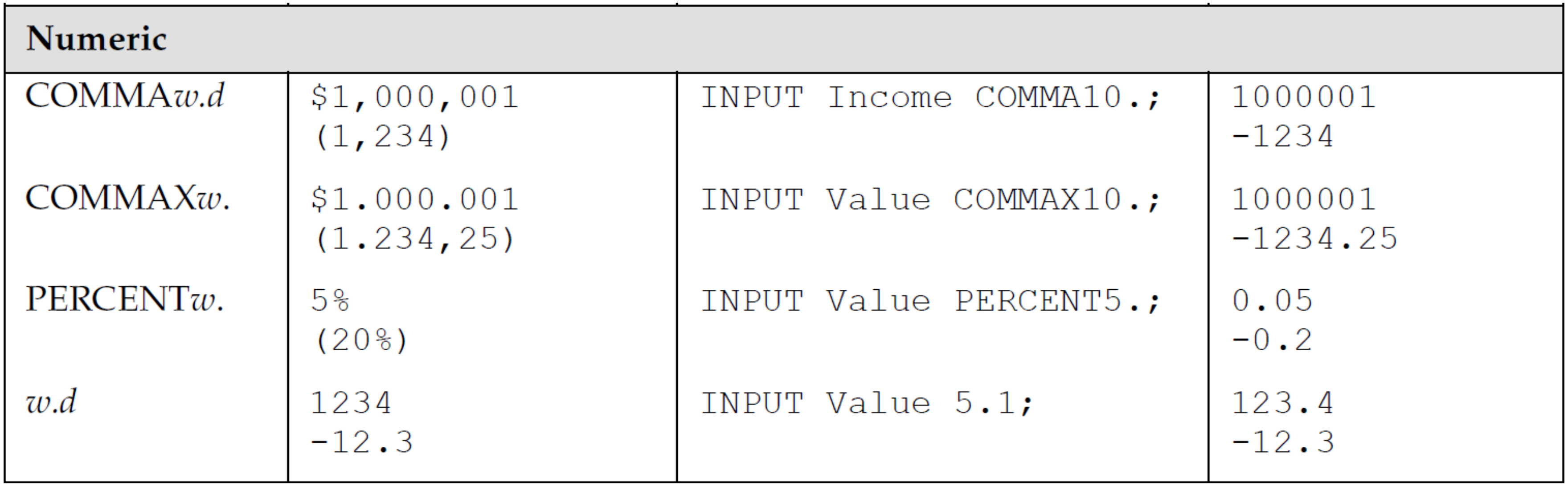
- The following table lists some commonly used input formats.



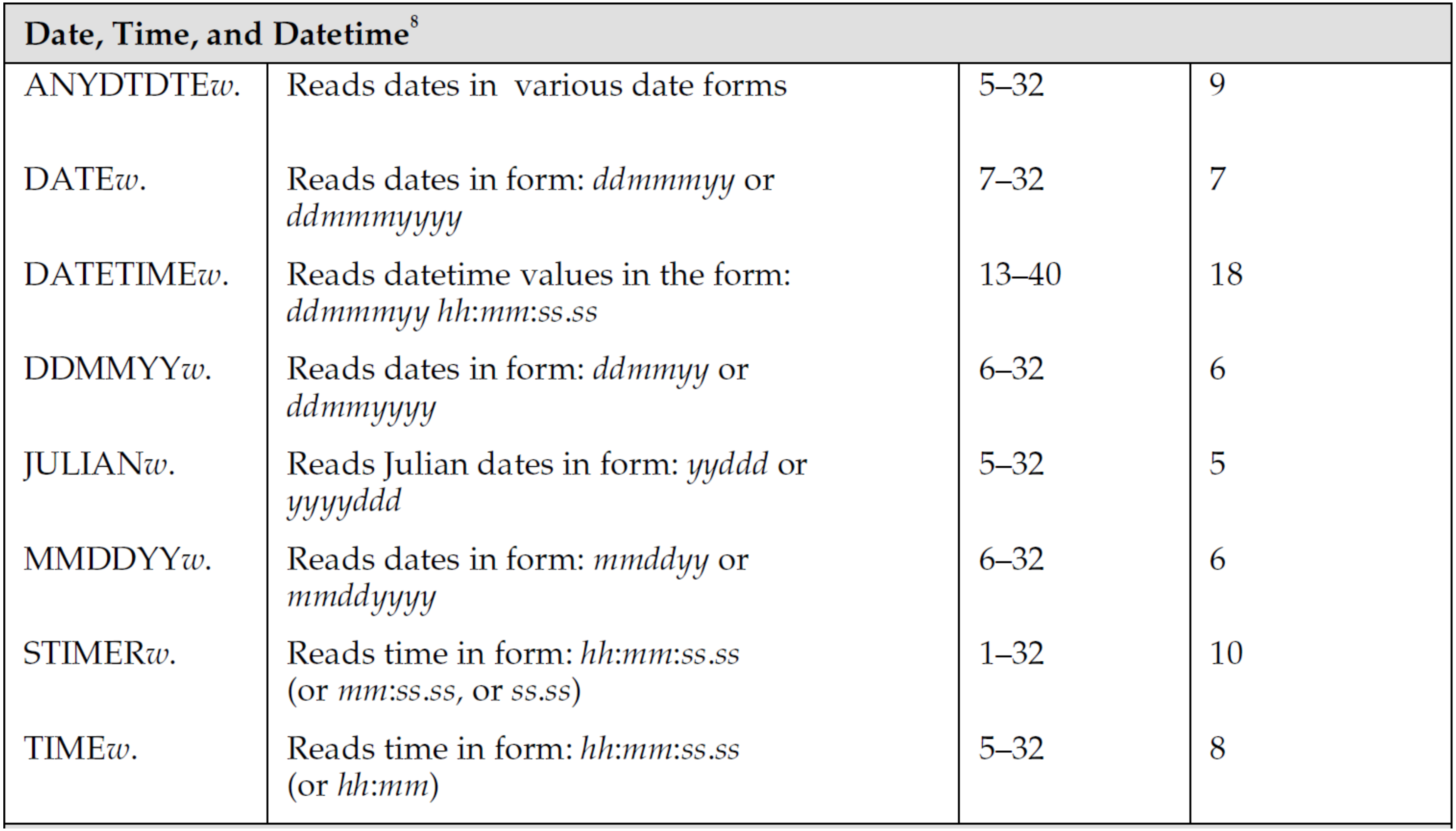
Informat	Definition	Width range	Default width
Character			
$CHAR*w*.	Reads character data—does not trim leading or trailing blanks	1–32,767	8 or length of variable
$UPCASE*w*.	Converts character data to uppercase	1–32,767	8
$*w*.	Reads character data—trims leading blanks	1–32,767	none
Numeric			
COMMA*w.d*	Removes embedded commas and $, converts left parentheses to minus sign	1–32	1
COMMAX*w*.	Like COMMA*w.d* but reverses role of comma and period	1–32	1
PERCENT*w*.	Converts percentages to numbers	1–32	6
w.d	Reads standard numeric data	1–32	none



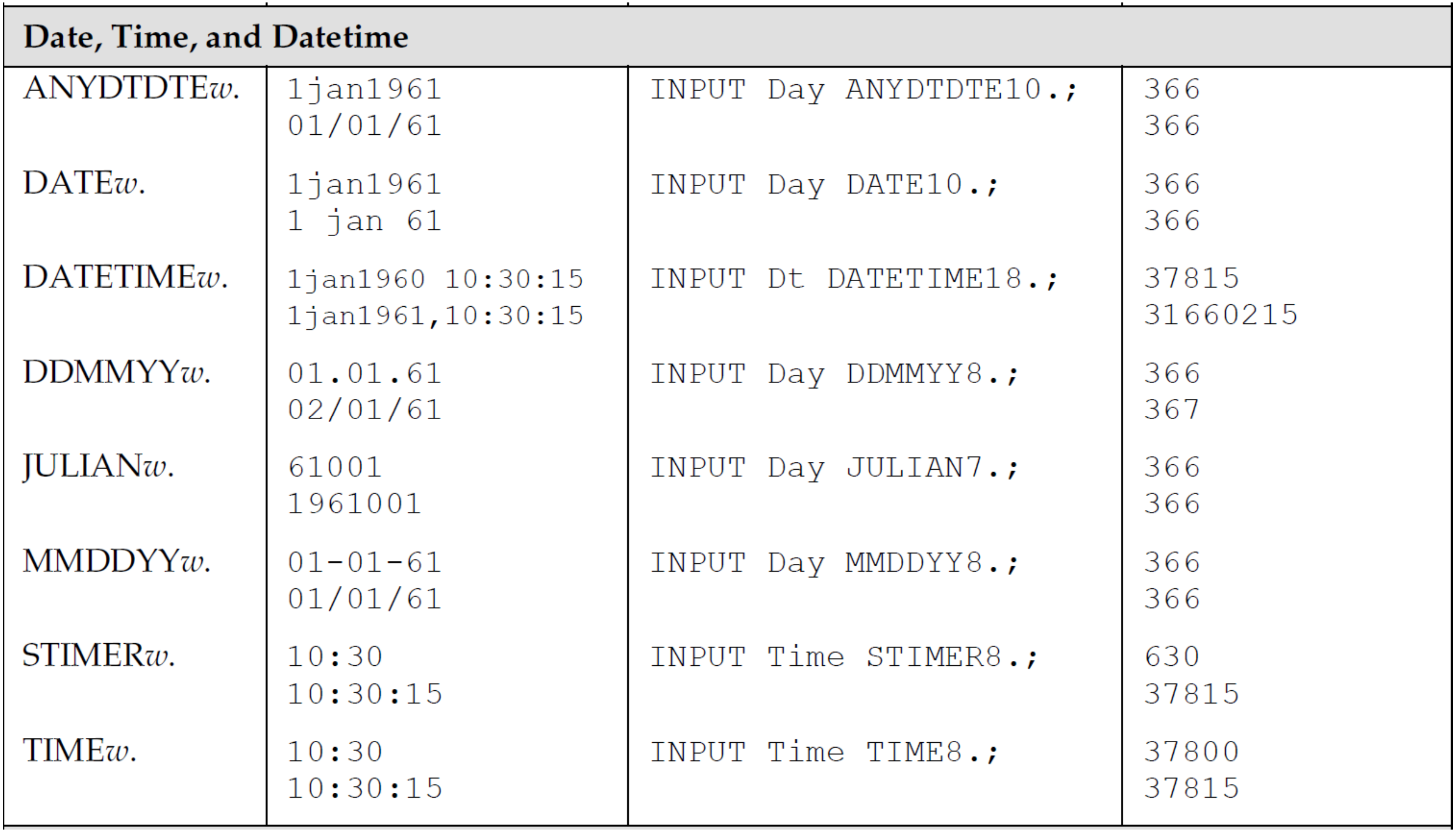
Informat	Input data	INPUT statement	Results
Character			
$CHAR*w*.	my cat my cat	INPUT Animal $CHAR10.;	my cat my cat
$UPCASE*w*.	my cat	INPUT Name $UPCASE10.;	MY CAT
$*w*.	my cat my cat	INPUT Animal $10.;	my cat my cat



Numeric			
COMMA*w.d*	$1,000,001 (1,234)	INPUT Income COMMA10.;	1000001 -1234
COMMAX*w*.	$1.000.001 (1.234,25)	INPUT Value COMMAX10.;	1000001 -1234.25
PERCENT*w*.	5% (20%)	INPUT Value PERCENT5.;	0.05 -0.2
w.d	1234 -12.3	INPUT Value 5.1;	123.4 -12.3



Date, Time, and Datetime[8]			
ANYDTDTE*w*.	Reads dates in various date forms	5–32	9
DATE*w*.	Reads dates in form: *ddmmmyy* or *ddmmmyyyy*	7–32	7
DATETIME*w*.	Reads datetime values in the form: *ddmmmyy hh*:*mm*:*ss*.*ss*	13–40	18
DDMMYY*w*.	Reads dates in form: *ddmmyy* or *ddmmyyyy*	6–32	6
JULIAN*w*.	Reads Julian dates in form: *yyddd* or *yyyyddd*	5–32	5
MMDDYY*w*.	Reads dates in form: *mmddyy* or *mmddyyyy*	6–32	6
STIMER*w*.	Reads time in form: *hh*:*mm*:*ss*.*ss* (or *mm*:*ss*.*ss*, or *ss*.*ss*)	1–32	10
TIME*w*.	Reads time in form: *hh*:*mm*:*ss*.*ss* (or *hh*:*mm*)	5–32	8



Date, Time, and Datetime			

ANYDTDTE*w*.	`1jan1961`
	`01/01/61`
		`INPUT Day ANYDTDTE10.;`	`366`
			`366`

DATE*w*.	`1jan1961`
	`1 jan 61`
		`INPUT Day DATE10.;`	`366`
			`366`

DATETIME*w*.	`1jan1960 10:30:15`
	`1jan1961,10:30:15`
		`INPUT Dt DATETIME18.;`	`37815`
			`31660215`

DDMMYY*w*.	`01.01.61`
	`02/01/61`
		`INPUT Day DDMMYY8.;`	`366`
			`367`

JULIAN*w*.	`61001`
	`1961001`
		`INPUT Day JULIAN7.;`	`366`
			`366`

MMDDYY*w*.	`01-01-61`
	`01/01/61`
		`INPUT Day MMDDYY8.;`	`366`
			`366`

STIMER*w*.	`10:30`
	`10:30:15`
		`INPUT Time STIMER8.;`	`630`
			`37815`

TIME*w*.	`10:30`
	`10:30:15`
		`INPUT Time TIME8.;`	`37800`
			`37815`

Example 8

The data file Pumpkin.dat represents the results from a local pumpkin-carving contest. Each line includes the contestant's name, age, type (carved or decorated), the date the pumpkin was entered, and the scores from each of five judges.

```
Alicia Grossman  13 c 10-28-2008 7.8 6.5 7.2 8.0 7.9
Matthew Lee       9 D 10-30-2008 6.5 5.9 6.8 6.0 8.1
Elizabeth Garcia 10 C 10-29-2008 8.9 7.9 8.5 9.0 8.8
Lori Newcombe     6 D 10-30-2008 6.7 5.6 4.9 5.2 6.1
Jose Martinez     7 d 10-31-2008 8.9 9.510.0 9.7 9.0
Brian Williams   11 C 10-29-2008 7.8 8.4 8.5 7.9 8.0
```

The following program reads these data.

```
* Read the file Pumpkin.dat using formatted input;
DATA contest;
INFILE 'D:\SAS\Pumpkin.dat';
INPUT Name $16. Age 3. +1 Type $1. +1 Date MMDDYY10.
     (Score1 Score2 Score3 Score4 Score5) (4.1);
RUN;
* Print the data set to make sure the file was read
correctly;
PROC PRINT DATA = contest;
     TITLE 'Pumpkin Carving Contest';
RUN;
```

- The variable Name has an informat of $16., meaning that it is a character variable of 16 columns wide.
- Variable Age has an informat of 3., is numeric, three columns wide, and has no decimal places.
- The +1 skips over one column.
- Variable Type is character, and it is one column wide.
- Variable Date has an informat MMDDYY10. and reads dates in the form 10-31-2013 or 10/31/2013, each 10 columns wide.
- The remaining variables, Score1 through Score5, all require the same informat, 4.1.
- By putting the variables and the informat in separate sets of parentheses, you only have to list the informat once.

Mixing input styles

- Each of the three major input styles has its own advantages.
- Column and formatted styles do not require spaces (or other delimiters) between variables and can read embedded blanks.
- Formatted style can read special data such as dates.
- SAS provides a flexible way that you can mix and match any of the input styles for your own convenience.

Example 9

Creating the data set SCORES using formatted input.

```
DATA SCORES;
INPUT @17 AGE 2. @1 NAME $15. @19 SEX $1. +1 GRADE 4.1;
CARDS;
CHAN CHI SHING  15M 70.5
CHEUNG TIN     *16M 67.2
LEUNG MAN SHAN  18F 72.0
MA YUEN LAN      .F 80.1
WU KEUNG        17M 75.3
RUN;
```

- In this example we use @n to control where to start reading the data.
- For example SAS starts reading AGE at column 17, and NAME at column 1.
- As $15. tells SAS explicitly that 15 columns are read for NAME, SAS sets the length for NAME to 15. Embedded blanks for NAME can also be read correctly.
- As there are no @n before GRADE in the INPUT statement, SAS starts reading GRADE from the current reading position (for this problem, it is column 20 because SAS reads SEX in column 19).
- Decimal points for GRADE are inserted according to the informat. If the informat 4.1 is replaced by 5.1, it will have no effect as SAS ignores blank.

Example 10

The following raw data contain information about U.S. national parks: name, state (or states as the case may be), year established, and size in acres:

```
Yellowstone           ID/MT/WY 1872    4,065,493
Everglades            FL 1934          1,398,800
Yosemite              CA 1864            760,917
Great Smoky Mountains NC/TN 1926         520,269
Wolf Trap Farm        VA 1966                130
```

```
* Read a data file NatPark.dat mixing input
styles;
DATA nationalparks;
    INFILE 'D:\SAS\NatPark.dat';
    INPUT ParkName $ 1-22 State $ Year @40
Acreage COMMA9.;
RUN;
PROC PRINT DATA = nationalparks;
    TITLE 'Selected National Parks';
RUN;
```

Notice that the variable `ParkName` is read with column style input, `State` and `Year` are read with list style input, and `Acreage` is read with formatted style input.

The ： modifier with an Informat

When data are not aligned in columns but we need additional instructions that only informats can provide, a ： modifier would be useful.

A ： modifier with an informat enables SAS to do the following:

- Treat the current field as a delimited field
- Apply an informat to the field, ignoring the width

Example 11

Compare the following input statements.

```
DATA Employee1;
     INPUT name $ salary:comma10. state $;
     * list input;
CARDS;
Ted $2.345 Georgia
Sam $222,345 Florida
RUN;

DATA Employee2;
     INPUT name $ salary comma10. state $;
     * formatted input;
CARDS;
Ted $2.345 Georgia
Sam $222,345 Florida
RUN;
```

Some `INFILE` options

A blank space is the default delimiter for list inputs. However, raw data are occasionally separated by different delimiter. E.g., CSV (comma-separated values) files.

There are two `INFILE` options to handle such cases.

- The `DLM` option specifies a delimiter to be used for list input
- The `DSD` option do the following:
  - Treat two consecutive delimiters as a missing value
  - Remove quotation marks from strings and treat any delimiter inside the quotation marks as a valid character
  - Set the default delimiter to a comma

Example 12

The programs below demonstrate the use of DLM and DSD options.

```
DATA kids;
   INFILE "D:\SAS\kids.dat" DSD;
   * INFILE 'D:\SAS\kids.dat' DLM=','; /*Does not work*/
   INPUT name $
         siblings
         bdate : mmddyy10.
         allowance : comma2.
         hobby1 : $10.
         hobby2 : $10.
         hobby3 : $10.;
RUN;
```

```
Chloe,,11/10/1995,,Running,Music,Gymnastics
Travis,2,1/30/1998,$2,Baseball,Nintendo,Reading
Jennifer,0,8/21/1999,$0,Soccer,Painting,Dancing
```

Note that two consecutive delimiters are treated as one, not as a missing value between the delimiters without the DSD option. Hence, the `INFILE` statement

```
INFILE 'D:\SAS\kids.dat' DLM=',';
```

does not work for reading `kids.dat`.

```
DATA kids_a;
   INFILE 'D:\SAS\kids_a.dat' DLM='/' DSD;
   INPUT name $
         siblings
         bdate : mmddyy10.
         allowance : comma2.
         hobby1 : $10.
         hobby2 : $10.
         hobby3 : $10.;
RUN;
```

```
Chloe/2/"11/10/1995"/$5/Running/Music/Gymnastics
Travis/2/"1/30/1998"/$2/Baseball/Nintendo/Reading
Jennifer/0/"8/21/1999"//Soccer/Painting/Dancing
```

Retrieving an existing SAS data set

We can use the SET statement to retrieve an existing permanent/temporary SAS data set for modification.

```
DATA [dsname];
     SET [[ref.]dsname];
     <We can put some other statements
    here to modify the SAS data set.>
RUN;
```

- In the `SET` statement, "`ref.dsname`" is the SAS data set that we want to retrieve. If "`ref`" is omitted, `WORK` is assumed.
- The `dsname` can be identical to that in the `DATA` statement. If no data set name is given, `SET` uses the data set most recently created in the present session.
- As the data are already stored in SAS format, we do not need to use `INPUT` statement to tell SAS how to read the data. The records in the SAS data set are read in from the existing file record by record.
- After reading a record, SAS executes the SAS modification statements that follow (so that the values in the record can be changed). SAS will read the next record only when the modification is completed.

Example 13

Suppose we have a SAS data set stored in the path `C:\RECORD` with filename `DATA01`. The following program creates a temporary copy of the SAS data set, and calls it ABC.

```
LIBNAME POPUL "C:\RECORD";
DATA ABC;
   SET POPUL.DATA01;
RUN;
```

Without using the `LIBNAME` statement, we can enter the data set name directly in the set statement.

```
DATA ABC;
   SET "C:\RECORD\DATA01";
RUN;
```