FAA

Fractal Academy of Analytics

The SAS System

An Introduction

What is SAS?

- Statistical Application Software
- The SAS system has a suite of products
- Each product associated with a set of functionalities
- Capable of efficiently handling very large data sets

Some products in the SAS System

- Core of the SAS System
  - The basic software to make SAS run
- Base SAS Software
  - SAS language, DATA step and Basic Procedures
- SAS/STAT
  - Procedures for various statistical analyses
- SAS/GRAPH
  - Procedures and options to create graphs
- SAS/ETS
  - Economic Time Series – Time Series Analysis
- SAS/OR
  - Operations Research – Optimization, LP etc
- SAS/EIS
  - Enterprise Information Systems – for OLAP models
- Enterprise Miner
  - Mining Package with various techniques
- SAS/Intr'net
  - Web based application and portal development
- Analyst/AF/FSP/Other Front End Based Features

Some points to note

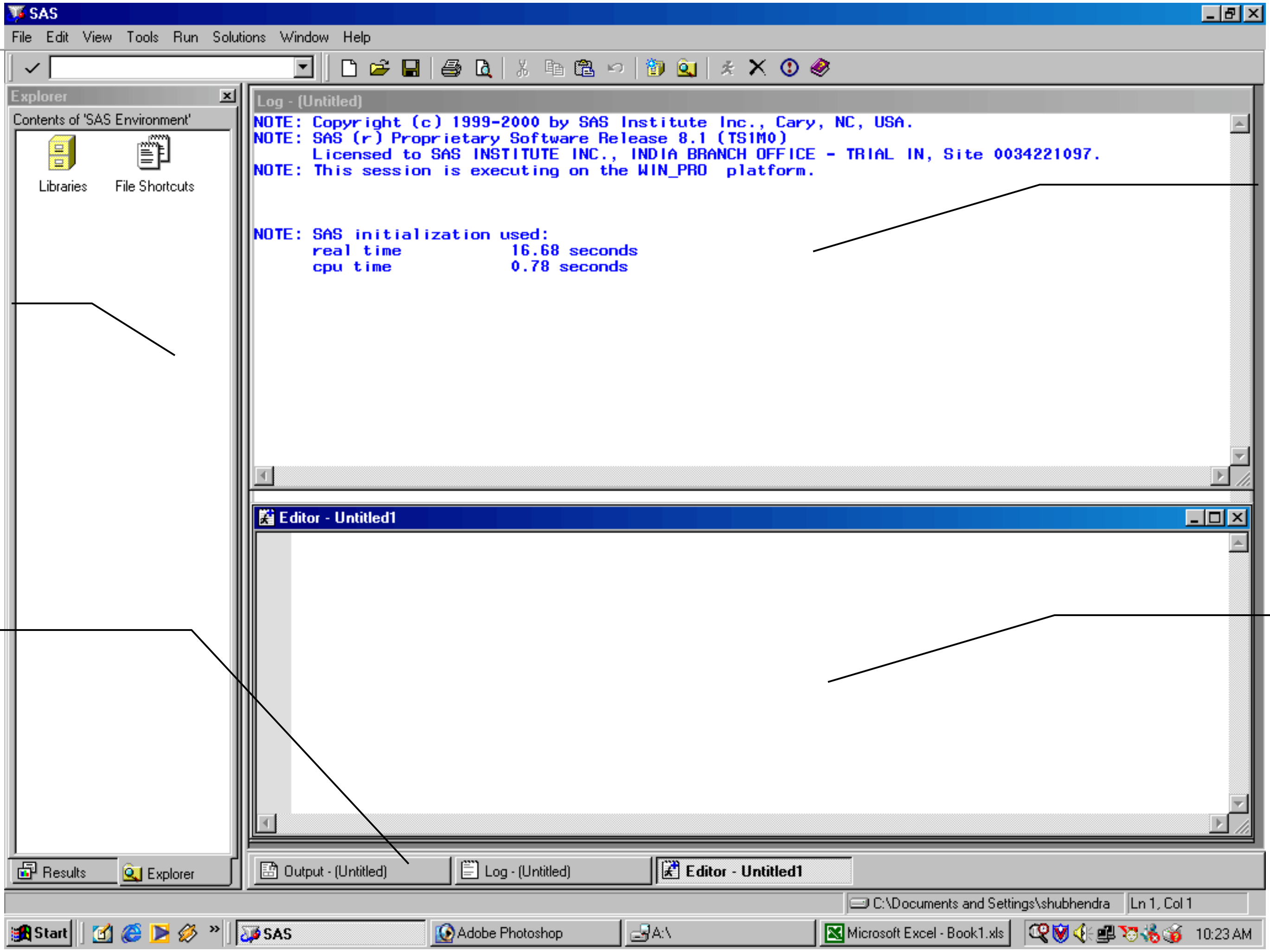
- The SAS language
  - SAS program is a sequence of statements executed in order
  - Layout of SAS programs
    - SAS statements can be in upper or lower case
    - Statements can continue on other line
    - Statements can be on the same line as other statements
    - Each statement must end with ;
  - Comments are to make programs more understandable
  - Commenting styles
    - Starts with * and ends with ;
    - Starts with /* and ends with */

Components of Base SAS

- SAS Data Sets
  - In-built format for storing data
  - Stores data as well as library information
  - Filename extensions for SAS Dataset
    - Version 7 or higher : ***.sas7bdat*** on Windows or Unix
    - Earlier versions : ***.sd2*** on Windows, ***.ssd01*** on Unix
  - Rules for SAS names
    - Names must be 32 characters or fewer in length
    - Names must start with a character or underscore
    - Names can contain only letters, numerals, or underscores (_) **No** %$*@#,;
    - Names are case insensitive

- The SAS Application
  - Program Editor, Log and Output Windows
  - Other windows

The SAS windows



```
NOTE: Copyright (c) 1999-2000 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 8.1 (TS1M0)
      Licensed to SAS INSTITUTE INC., INDIA BRANCH OFFICE - TRIAL IN, Site 0034221097.
NOTE: This session is executing on the WIN_PRO  platform.



NOTE: SAS initialization used:
      real time           16.68 seconds
      cpu time            0.78 seconds
```

Explorer

Log

Output

Program Editor

Components of the SAS language

- DATA step
  - Creates one or more SAS datasets
  - From existing SAS datasets or external sources
- PROC step
  - Performs various operations on SAS datasets
  - Usually does not operate on external sources
- Open Code
  - Any statement which is not part of a DATA step or PROC step

The **LIBNAME** statement

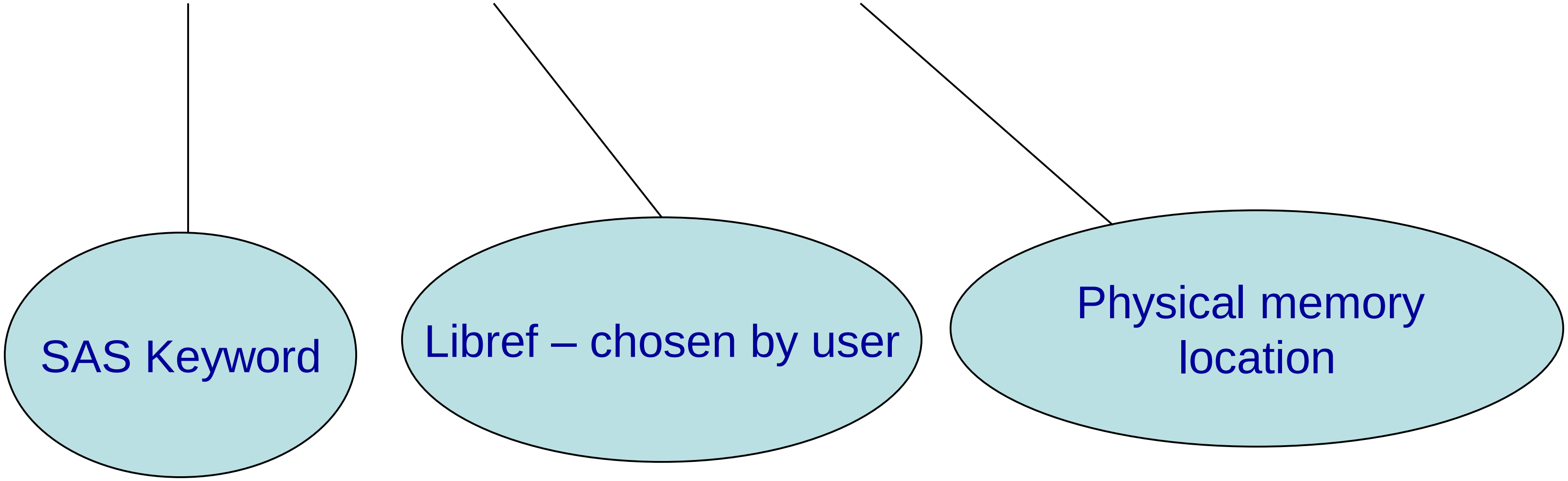
- Library
  - Physical location with SAS data sets
- Library Reference or Libref
  - Alias used by SAS to reference a Library
- **LIBNAME** statement
  - Assigns a Libref to a Library
  - Occurs in Open Code
- Temporary datasets are stored in a Library called “WORK”
  - Physically a temporary folder is created which gets erased after the SAS session is closed

Syntax for **LIBNAME**

LIBNAME *<libref>* “*<location>*”***<OPTIONS>;***

Examples

LIBNAME *TRAINING* “C:\TRAINING***” ;***



SAS Keyword

Libref – chosen by user

Physical memory
location

Introduction to SAS Data Sets

- Can be temporary or permanent
- Temporary datasets
  - Are usable only while the session is active
  - Are stored in a temporary folder
    - Deleted when the SAS session is closed
  - Are referred by a “one-level name”
- Permanent datasets
  - Are usable also after SAS session is closed
  - Are stored in a permanent physical location
    - Permanent physical location unaffected by SAS session
  - Are referred by a “two-level name”

SAS Datasets

- Permanent Data Sets – Two level names
  - <libref>.<data name>
  - Example: *TRAINING.*TESTDATA

- Temporary Data Sets – One level name
  - <libref>.<data name>
  - Example: TESTDATA
  - Note that this is equivalent to *WORK.*TESTDATA
    - It is NOT a two level name

- All DATA steps and most procedures reference SAS data sets by the rules specified above

The DATA Step

- Creates and manipulates SAS data sets
- Can be used for creating SAS data sets from
  1. Data entry in the program editor
  2. Flat files in a host of different formats
  3. Existing SAS Data sets
- Can be used to output data directly to flat files
- Can be used to create multiple data sets in one go

Notes

1. The DATA step cannot create SAS Data sets from binary files (like Excel) directly. Procedures as well as front-end features are available for carrying out such operations.
2. Similar for outputting SAS Data sets to binary files

The basic syntax of DATA step

DATA <DATA1> [<DATA2> … <DATAn>];

…

…

…

RUN;

IMPORTANT NOTES:

1. Every SAS statement ends with a semi-colon
2. Every block of SAS code (DATA step or PROC step) ends with **RUN**
3. SAS code is free format, i.e. there are no restrictions on where in a line and column the code can begin and end

Methods for getting your data into the SAS system

- Entering data directly into SAS dataset
- Creating SAS datasets from raw files
  - Using Data step
  - Using Import Procedure
- Converting other software’s data files into SAS datasets
- Reading other software’s data files directly

Reading Instream Data

- To read instream data, use a **DATALINES statement** and immediately preceding the data lines
- a **null statement** (a single semicolon) to indicate the end of the input data.

```
data clinic.stress;
  input ID 1-4 Name $ 6-25 RestHr 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
   datalines;
   .
data lines go here
   .
;
```

- If your data contains semicolons, use the **DATALINES4** statement plus a null statement that consists of four semicolons (**;;;;**) to indicate the end of the input data.

Using Data step for Internal raw data

- Internal raw data
  - Datalines or Cards to indicate that the data is internal

```
data cities;
input City $ Rank ;
datalines;
Mumbai 1
Delhi   2
Chennai 3
Calcutta 4
;
run ;
```

Using Data step for External raw data

- External raw data
  - – Infile statement to tell SAS the filename and the path

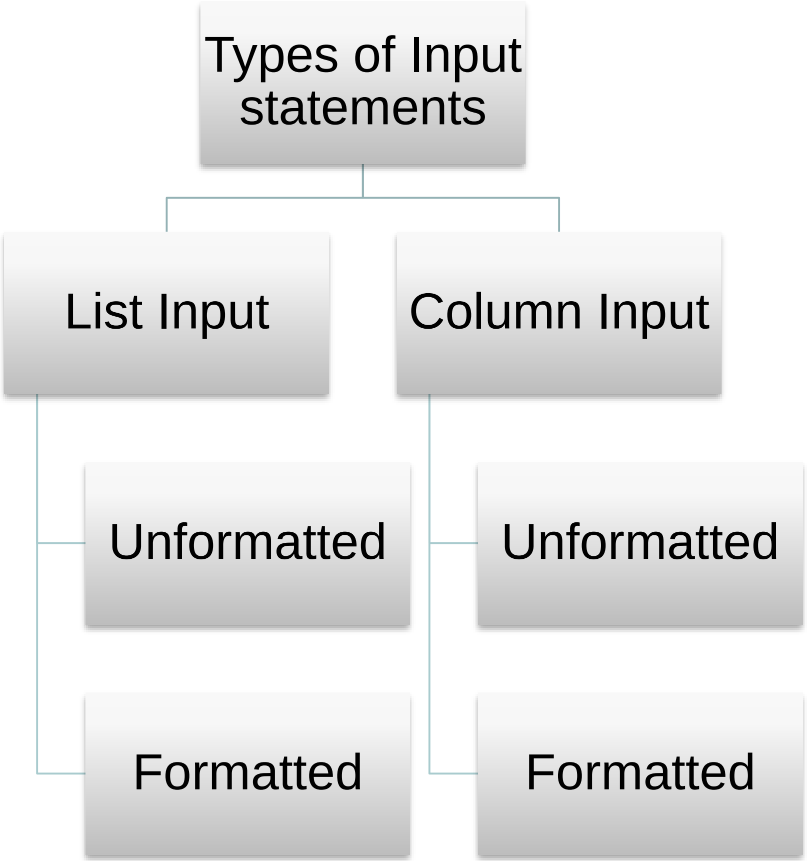
Text in the external file

```
Mumbai 1
Delhi    2
Chennai 3
Calcutta 4
```

```
data cities;
infile "C:\training\sample1.txt”     ;
input City $ Rank ;
run ;
```

```
NOTE: The infile "C:\training\sample1.txt" is:      File Name=C:\training\sample1.txt,
RECFM=V, LRECL=256
NOTE: 4 records were read from the infile "C:\training\sample1.txt".
      The minimum record length was 8.
      The maximum record length was 10.
NOTE: The data set WORK.CITIES has 4 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.25 seconds
      cpu time            0.11 seconds
```

Different types of input



Types of Input statements

List Input

Unformatted

Formatted

Column Input

Unformatted

Formatted

Reading Raw data separated by spaces

- List Input

```
data runners;
input name $ surname $ age runtime1 runtime3 ;
datalines;
Scott  A 15
23.3 21.5
Mark .  13 25.2 24.1
;
run ;
```

```
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.RUNNERS has 2 observations and 5 variables.
```

- All missing data must be indicated by a period
- All values are separated by at least one space
- Character data are eight characters or fewer
- Should not have embedded spaces

Reading Raw data separated by spaces

```
data runners;
input name $ surname $ age runtime1
runtime2 ;
datalines;
Scott  A 15
22.0 21.9
Mark .  13 25.2 24.1
Jon K 13 25.1
Michael M 14 12 .
;
run ;
```

```
data runners;
input name $ surname $ age runtime1
runtime2 ;
datalines;
Scott  A 15
22.0 21.9 Mark .  13 25.2 24.1
Michael M 14 12 .
;
run ;
```

```
NOTE: Invalid data for runtime2 in line 228 1-7.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+-
228        Michael M 14 12 .
name=Jon surname=K age=13 runtime1=25.1 runtime2=. _ERROR_=1 _N_=3
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.RUNNERS has 4 observations and 5 variables.
```

Reading Missing Values

- Reading Missing Values at the End of a Record ?

- The MISSOVER option in the INFILE statement
  - The MISSOVER option prevents SAS from going to another record
  - when using list input, it does not find values in the current line for all the INPUT statement variables
  - At the **end** of the current record, values that are expected but not found are set to missing.
  - The MISSOVER option works only for missing values that occur at the **end** of the record.
- *Example :*

```
data perm.survey;
infile credit missover;
input Gender $ Age Bankcard FreqBank        Deptcard FreqDept;
run;
```

Reading Missing Values

- Reading Missing Values at the Beginning /Middle of a Record
- The DSD Option
  - The DSD option changes how SAS treats delimiters when list input is used.
  - sets the default delimiter to a comma
  - treats two consecutive delimiters as a missing value
  - removes quotation marks from values.
- You can also use the DSD and DLM= options to read fields that are delimited by blanks.
- *Example :*

```
data perm.survey;
infile credit dsd missover dlm=‘ ’;
input Gender $ Age Bankcard FreqBank        Deptcard FreqDept;
run;
```

Reading Delimited Files with the DATA step

```
data sample;
infile 'c:\training\sample6.txt' dlm=',' dsd ;
input x y ;
run ;
```

Data in the sample6.txt file

```
123, 1,
23,,,2,
45, ,3,
67, 4,
```

What is the output with and without **dsd** option ?

Is the comma at the end needed?

Data in the sample7.txt file

```
123, 1,3,5
23,,,2,
45, ,3,
67, 4,
```

What is the output with and without **dsd** option ?

Controlling input with options in the Infile statement

```
infile ‘C:\training\sample.txt' missover firstobs=7 obs = 100 ;
```

- **Firstobs =**
  - tells SAS where to start reading the file
- **Obs =**
  - tells SAS to stop reading when it gets to that line in the raw data
  - it does not necessarily correspond to the number of observations and it really is “lastobs”
- **Missover**
  - prevents SAS from going to a new input line when it does not find values in the current line for some of the variables declared in the input statement.
  - With the MISSOVER option, when SAS reaches the end of the current record, variables without any values assigned are set to missing.

Creating SAS Data Sets from Raw Data

- ***Raw Data Files***

  A raw data file is an external text file whose records contain data values that are organized in fields

- ***Steps to Create a SAS Data Set***

To Do This	Use This SAS Statement	Example
Reference a SAS data library	LIBNAME statement	libname libref 'SAS-data-library';
Reference an external file	FILENAME statement	filename tests 'c:\users\tmill.dat';
Name a SAS data set	DATA statement	data clinic.stress;
Identify an external file	INFILE statement	infile tests obs=10;
Describe data	INPUT statement	input ID 1-4 Age 6-7 ...;
Execute the DATA step	RUN statement	run;
List the data	PROC PRINT statement	proc print data=clinic.stress;
Execute the final program step	RUN statement	run;

Verifying the Data

- Whenever you use the DATA step to read raw data,
  - Write the DATA step using OBS= option in the INFILE statement.(e.g. infile tests obs=10;)
  - Submit the DATA step.
  - Check the log for messages.
  - View the resulting data set.
  - Remove the OBS= option and re-submit the DATA step.
  - Check the log again.
  - View the resulting data set again.

- ***Creating and Modifying Variables***

  General form, assignment statement:

  *variable=expression**;***

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

Reading Free-Format Data

- Reading a Range of Variables
  - input IDnum $ **Ques1-Ques5**;
  - input Age **(Store1-Store3) ($);**

- **Limitations of List Input**
  - Although the width of a field can be greater than eight columns, both character and numeric variables have a default length of 8. Character values that are longer than eight characters will be truncated.
  - Data must be in standard numeric or character format.
  - Character values cannot contain embedded delimiters.
  - Missing numeric and character values must be represented by a period or some other character.

Reading Raw Data in Fixed Fields

- Column Input Features
  - It can be used to read character variable values that contain embedded blanks
  - No placeholder is required for missing data. A blank field is read as missing and does not cause other fields to be read incorrectly.
  - Fields or parts of fields can be re-read.
  - Fields do not have to be separated by blanks or other delimiters.

- **Standard Numeric Data**

  Standard numeric data values can contain only
  - numbers
  - decimal points
  - numbers in scientific, or E, notation (23E4)
  - minus signs and plus signs.

Reading Raw Data in Fixed Fields

General form, INPUT statement using column input:

INPUT *variable ;*<$> *startcol-endcol . . .*

where

- *variable* is the SAS name that you assign to the field
- the dollar sign ($) identifies the variable type as character (if the variable is numeric, then nothing appears here)
- *startcol* represents the starting column for this variable
- *endcol* represents the ending column for this variable.

```
filename exer 'c:\users\exer.dat';
data exercise;
  infile exer;
  input ID $ 1-4 Age 6-7 ActLevel $ 9-12 Sex $ 14;
run;
```

Using Informats

- An informat is an instruction that tells SAS how to read raw data.

PERCENTw.d	**DATEw.**	**NENGOw.**
$BINARYw.	DATETIMEw.	PDw.d
$VARYINGw.	HEXw.	PERCENTw.
$w.	JULIANw.	TIMEw.
COMMAw.d	MMDDYYw.	w.d

- Note that
  - each informat contains a *w* value to indicate the width of the raw data field
  - each informat also contains a period, which is a required delimiter
  - for some informats, the optional *d* value specifies the number of implied decimal places
  - informats for reading character data always begin with a dollar sign ($).

Reading Date and Time Values

- How SAS Stores **Date** Values ?
  - When you use a SAS informat to read a date, SAS converts it to a **numeric date value**. A SAS date value is the number of days from January 1, 1960, to the given date.
- How SAS Stores **Time** Values ?
  - SAS stores time values similar to the way it stores date values. A SAS time value is stored as the number of seconds since midnight.
- A SAS **datetime** is a special value that combines both date and time information. A SAS datetime value is stored as the number of seconds between midnight on January 1, 1960, and a given date and time.

Date and Time Informats

- MMDDYY*w*. Informat
  - Reads *mmddyy* or *mmddyyyy*

Date Expression	SAS Date Informat
101599	MMDDYY6.
10/15/99	MMDDYY8.
10 15 99	MMDDYY8.
10-15-1999	MMDDYY10.

- DATE*w*. Informat
  - Reads *ddmmmyy* or *ddmmmyyyy*

Date Expression	SAS Date Informat
30May00	DATE7.
30May2000	DATE9.
30-May-2000	DATE11.

Date and Time Informats

- TIME*w*. Informat
  - *Reads hh:mm:ss.ss*
  - where
    - *hh* is an integer from 00 to 23, representing the hour
    - *mm* is an integer from 00 to 59, representing the minute
    - *ss.ss* is an optional field that represents seconds and hundredths of seconds.

Time Expression	SAS Time Informat
17:00:01.34	TIME11.
17:00	TIME5.
2:34	TIME5.

- Five is the minimum acceptable field width for the TIME*w*. informat.

Date and Time Formats

- The WEEKDATE*w*. Format
  - The WEEKDATE*w*. format writes date values in the form *day-of-week*, *month-name dd*, *yy* (or *yyyy*).

FORMAT Statement	Result
format datein weekdate3.;	Mon
format datein weekdate6.;	Monday
format datein weekdate17.;	Monday, Apr 5, 99
format datein weekdate21.;	Monday, April 5, 1999

- The WORDDATE*w*. Format
  - The WORDDATE*w*. format is similar to the WEEKDATE*w*. format, but it does not display the day of the week or the two-digit year values.

FORMAT Statement	Result
format datein worddate3.;	Apr
format datein worddate5.;	April
format datein worddate14.;	April 15, 1999