Structure of a SAS Program

- SAS programs consist of SAS statements
  - General Form: `SAS_keyword operand;`
    e.g. `PROC ANOVA DATA=CORN;`
- SAS statements are interpreted and executed in their order of appearance
- However, certain blocks of statements called *steps*, which define one of two basic activities, are executed as groups
  - **DATA** step: statements leading to the creation of a SAS data set
  - **PROC** step: statements needed to tell a SAS procedure to perform a statistical analysis
- A SAS program consists of several logically related **DATA** and **PROC** steps
**SAS Program (Example A0)**

```sas
data oranges;
  input Variety $ Flavor Texture Looks;
  Total=Flavor+Texture+Looks:
datalines;
navel 9 8 6
temple 7 7 7
valencia 8 9 9
mandarin 5 7 8
;
proc sort data=oranges;
  by descending Total;
run;
proc print data=oranges;
  title 'Taste Test Results for Oranges';
run;
```
Some Syntax Rules

- **SAS names**: Names you give to SAS variables, SAS data sets etc.
  - Max. of 8 characters long
  - Must begin with an alpha character
  - Can use the notation Q1-Q12 to name a list of variables Q1, Q2, ..., Q12

- **SAS statements**:
  - Begin and end anywhere in a line
  - Must end with a ';'
  - More than one statement may appear in a line
  - A statement may take more than one line
  - Items in SAS statements must be separated by at least one blank (unless already separated by a SAS symbol)
    - e.g., x = y;  x =y;  x= y;  x=y;
SAS Data Sets

- Consist of *data values* arranged in a rectangular array
- Columns of this array represent the *SAS variables*
- Rows represent *observations*
- In addition to the data values, the *attributes* associated with each variable are also kept in the SAS data set
- SAS data set has *a special structure* which is different from data sets you create using an editor, such as an ASCII data file
- Most SAS *procedures* can access only SAS data sets
- SAS data sets are usually created in a data step and may be *temporary* or *permanent*
A SAS data set is a rectangular table of rows and columns.

In a SAS data set, all variables must have a name, type, and length. A variable’s type is either character (string) or numeric. The type plays a role in determining the length.
A **format** is used to control how values are displayed. The table below is what you might see in the printed output. Formats do not affect how values are stored internally (see next slide).

<table>
<thead>
<tr>
<th>Customer_Name</th>
<th>Customer_Age</th>
<th>Order_Date</th>
<th>Total_Retail_Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Klisurich</td>
<td>38</td>
<td>01/11/2003</td>
<td>$16.50</td>
</tr>
<tr>
<td>Sandrina Stephano</td>
<td>28</td>
<td>01/15/2003</td>
<td>$247.50</td>
</tr>
<tr>
<td>Dianne Patchin</td>
<td>28</td>
<td>01/20/2003</td>
<td>$28.30</td>
</tr>
<tr>
<td>Wendell Summersby</td>
<td>43</td>
<td>01/28/2003</td>
<td>$32.00</td>
</tr>
<tr>
<td>Duncan Robertshawe</td>
<td>63</td>
<td>02/27/2003</td>
<td>$63.60</td>
</tr>
<tr>
<td>Najma Hicks</td>
<td>21</td>
<td>03/02/2003</td>
<td>$234.60</td>
</tr>
</tbody>
</table>

Format: MMDDYYYY  
Width: 10  
Stored value: 15766

Format: DOLLAR  
Width: 8  
Decimal Places: 2  
Stored value: 234.60
SAS Data Set

When SAS data values are stored internally in the computer memory only as character values or numeric values.

<table>
<thead>
<tr>
<th>Customer_Name</th>
<th>Customer_Age</th>
<th>Order_Date</th>
<th>$</th>
<th>Total_Retail_Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Klisurich</td>
<td>38</td>
<td>15716</td>
<td></td>
<td>16.50</td>
</tr>
<tr>
<td>Sandrina Stephano</td>
<td>28</td>
<td>15720</td>
<td></td>
<td>247.50</td>
</tr>
<tr>
<td>Dianne Patchin</td>
<td>28</td>
<td>15725</td>
<td></td>
<td>28.30</td>
</tr>
<tr>
<td>Wendell Summersby</td>
<td>43</td>
<td>15733</td>
<td></td>
<td>32.00</td>
</tr>
<tr>
<td>Duncan Robertshawe</td>
<td>63</td>
<td>15763</td>
<td></td>
<td>63.60</td>
</tr>
<tr>
<td>Najma Hicks</td>
<td>21</td>
<td>15766</td>
<td></td>
<td>234.60</td>
</tr>
</tbody>
</table>

Character values are 1 to 32,767 characters (bytes) long.

Numeric values are 8 bytes of floating point storage:
- Numeric
- Currency
- Date (days from 01JAN1960)
- Time (seconds from midnight)
A **format** is an instruction that SAS uses to convert internal data values to the written appearance.

The FORMAT statement can be used to associate standard SAS formats (or user-written formats) with variables.

You use a FORMAT statement in the DATA step to *permanently* associate a format with a variable.

You can use a FORMAT statement in some PROC steps, to *temporarily* associate a format with a variable.

The FORMAT procedure enables you to define your own formats for variables (user-written formats)
Some Common Formats

- **w.d**
  
  Example: Internal value **1234.5** Format **8.2**
  Output **b1234.50**

- **w.**
  
  Example: Internal value **1234** Format **6.**
  Output **bb1234**

- **$w.**
  
  Example: Internal value **Mary** Format **$5.**
  Output **Mary#**

- **mmddyyw.**
  
  Example: Internal value **18537** Format **mmddyy8.**
  Output **10/02/10**

- **dollarw.d**
  
  Example: Internal value **123.45** Format **dollar8.2.**
  Output **bb$123.45**

*Note: b and # symbols in above examples denote blanks*
Missing Values

If a data value is not present for a variable in a particular observation, it is considered *missing*.

- A missing character value is displayed as a blank.
- A missing numeric value is displayed as a period.
SAS Data Step

- Begins with the statement
  - `DATA name ;`
- Followed by **one** of these statements:
  - `INPUT ;`
  - `SET ;`
  - `MERGE ;`
  - `UPDATE ;`
- A SAS data step is (usually) used to create a new **SAS data set** from
  - external data (using an `INPUT` statement)
  - another SAS data set (using a `SET` statement)
- **SAS program statements** are used in a SAS data step to modify input data, if necessary
Flow of Operations: Data Step

START

SAS reads a line of data

SAS carries out program statements for this data line and creates new observation

SAS adds observation to new data set

If no more lines of data to input SAS closes the data set and goes on to next DATA or PROC statement

SAS returns for a new line of data
data oranges;
  input Variety $ Flavor Texture Looks;
  Total=Flavor+Texture+Looks;
datalines;
navel 9 8 6
  temple 7 7 7
  valencia 8 9 9
  mandarin 5 7 8
run;
;
SAS Variables

- **Data values** stored in SAS data sets are of two ‘type’s
  - Character
  - Numeric
- **SAS variables** that contain these values have several *attributes* associated with them
  - Name
  - Type
  - Length (in bytes)
  - Relative position
  - Informat
  - Format
  - Label
Simple INPUT Statements

- **List Input**
  
  ```
  INPUT    ID   SEX   $   AGE   WEIGHT ;
  1342       F      27     121.2
  
  INPUT    SCORE1-SCORE4 ;
  63.1      94    87.5    72
  ```

- **Formatted Input**
  
  ```
  INPUT  ID  4.  STATE  $2.  FERT  5.2  PERCENT  3.2 ;
  0001IA_ _504089
  
  INPUT  @10  ITEM  $4.  +5  PRICE  6.2;
  xxxxxxxxxxxR2D2xxxxxx_91350
  
  INPUT (ID  SEX  AGE  WT  HT) (3.  $1.  2.  2*5.1);
  123M21_1650_ _721
  ```

- **Column Input**
  
  ```
  INPUT  ID 1-4  STATE  $  5-6  FERT  7-12
  PERCENT  13-15  .2 ;
  0001IA_ _5.04_89
  ```
SAS Program Statements

\[ Y_1 = X_1 + X_2^{**2}; \]
\[ Y_2 = \text{ABS}(X_3); \]
\[ Y_3 = \text{SQRT}(X_4 + 4.0 \times X_5^{**2}) - X_6; \]
\[ X_7 = 3.14156 \times \text{LOG}(X_7); \]

\[ \text{IF INCOME = . THEN DELETE ;} \]
\[ \text{IF STATE = 'CA' | STATE = 'OR' THEN} \]
\[ \quad \text{REGION = 'PACIFIC COAST';} \]
\[ \text{IF SCORE < 0 THEN SCORE = 0;} \]

\[ \text{IF SCORE < 80 THEN WEIGHT = .67;} \]
\[ \text{ELSE WEIGHT = .75;} \]

\[ \text{WEIGHT = (SCORE < 80) \times .67 + (SCORE \geq 80) \times .75;} \]

\[ \text{IF SCORE < 80 THEN DO;} \]
\[ \quad \text{WEIGHT = .67;} \]
\[ \quad \text{RATE = 5.70;} \]
\[ \quad \text{END;} \]
\[ \text{ELSE DO;} \]
\[ \quad \text{WEIGHT = .75;} \]
\[ \quad \text{RATE = 6.50;} \]
\[ \quad \text{END;} \]
IF 6.5 <= RATE <= 7.5 THEN GO TO USEIT;

Add SAS statements here to calculate a new value for RATE:

USEIT: COST = HOURS * RATE;

DATA;
INPUT X 1 – X 5;
X6 = (X4 + X5) / 2;
DROP X4 X5;
DATALINES;
;
;

data iterate;
  input x;
  do i=1 to 10;
    y = x * normal(0);
    output;
  end;
datalines;
5
-10
2500
;
Order of Evaluating Expressions

- **Rule 1**: Expressions within *parenthesis* are evaluated first

- **Rule 2**: Higher *priority* operators are performed first
  
  - **Group I**: \( **, + \text{(prefix)}, - \text{(prefix)}, ^\text{(NOT)} \)
  - **Group II**: \(*, / \)
  - **Group III**: \(+ \text{(infix)}, - \text{(infix)} \)
  - **Group IV**: \(| | \)
  - **Group V**: \(<, <=, =, ^=, >=, >, ^>, ^< \)
  - **Group VI**: \& \text{(AND)}
  - **Group VII**: \| \text{(OR)}

- **Rule 3**: For operators with the same priority, the operations take place from left to right of the expression (except for Group I operators, which are executed right to left.)
A SAS function is a routine that returns a value that is determined from specified arguments.

General form of a SAS function:

```
function-name(argument1,argument2, ...)  
```

- **Examples:**

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

```
ave=mean(flavor,texture,looks)
```

```
id=substr(item,1,2)
```
Using SAS Functions

- SAS functions can do the following:
  - perform arithmetic operations
  - compute sample statistics (for example: sum, mean, and standard deviation)
  - manipulate SAS dates
  - process character values
  - perform many other tasks

📝 Sample statistics functions ignore missing values.
Example A4

data group1;
    input Age @@;
datalines;
1 3 7 9 12 17 21 26 30 32 36 42 45 51;
data group2;
set group1;
    if 0<=Age<10 then Agegroup=0;
    else if 10<=age<20 then Agegroup=10;
    else if 20<=age<30 then Agegroup=20;
    else if 30<=age<40 then Agegroup=30;
    else if 40<=age<50 then Agegroup=40;
    else if age >=50 then Agegroup=50;
run;
proc print;
run;

data group3;
set group1;
    Agegroup=int(Age/10)*10;
run;
proc print;
run;
The Proc Step

A simple proc step:

PROC PRINT;

Prints a listing of

- the most recently created data set
- all variables contained in the data set
- all observations in the data set

General form of the proc step:

PROC proc-name options-list;

- procedure information statements
- variable attribute statements

Examples of options:

DATA=NEW STDERR KENDALL

Examples of procedure information statements:

VAR ... ;
BY ... ;
CLASS ... ;
MODEL ... ;
OUTPUT ... ;

Examples of variable attribute statements:

DROP ... ;
FORMAT ... ;
LABEL ... ;
The BY Statement

**BY variables_list;**

- Allows the **processing of subsets** of the same data set in one proc step
- Procedure is **executed repeatedly** on each subset of observations defined by values of variables listed in the BY statement
- **Subset of observations** to be processed together is defined as those having the same values for the specified variables
- These observations must **occur together** (i.e., contiguously) in the data set
- Usually the given data set to be processed in **subset must be rearranged** so that the observations are in the required groupings
- This rearrangement is most easily achieved using **PROC SORT**
Simple Uses of Proc SORT

PROC SORT;
BY State;
;
- Input data set is the one created most recently
- Output data set replaces the input data set
- Observations will be arranged in the increasing order of the values for State
- More importantly, the observations with the same value for State will appear together

PROC SORT;
BY State City;
;
- Observations will be arranged in the increasing order of State
  Within each State, the observations will be arranged in the increasing order of CITY
- Observations with the same values for State and City will be grouped together
PROC SORT DATA=survey ;
   BY Gender Income ;
PROC PRINT ;
   BY Gender ;
PROC MEANS MEAN VAR ;
   BY Gender Income ;
   VAR Age Food Rent ;
   :

Output 1   GENDER=F
   Listing of observations in the increasing order of INCOME
   GENDER=M
   Listing of observations in the increasing order of INCOME
Simple Uses of Proc SORT (cont)

Output 2

GENDER=F    INCOME=1

Means and variances of each of the variables AGE etc.

GENDER=F    INCOME=2

...          

GENDER=F    INCOME=3

...          

GENDER=M    INCOME=1

...          

GENDER=M    INCOME=2

...          

GENDER=M    INCOME=3
Examples of SAS Procedures

PROC MEANS MEAN VAR ;
   VAR Age Income ;
Other Options:  DATA= , NOPRINT, MAXDEC=, N, MISS,
                MEAN, STD, MIN, MAX, RANGE, SUM, VAR, STDERR,
                CV, T, PRT  etc.,

PROC CORR ;
   VAR  Height  Weight;
   WITH BP Oxygen;
Some Options:  DATA= , PEARSON, SPEARMAN, BEST= ,
               NOSIMPLE, NOPRINT, etc.,

PROC UNIVARIATE ;
   CLASS  County;
   VAR Acreage Rainfall;
   OUTPUT OUT= new MEAN= Ave1 Ave2 VAR= V1 V2 ;
Some Options:  DATA= , NOPRINT, PLOT, NORMAL,
               PCTLDEF=,VARDEF=, ALPHA= , CIBASIC(TYPE= ALPHA=) ,
               Muo=, TRIM=(TYPE= ALPHA=), etc.

Some Keywords for OUTPUT:  N, NMISS, NOBS, MEAN,
SUM, SD, VAR, SKEWNESS, KURTOSIS, MAX, MIN,
RANGE, Q3, MEDIAN, Q1, P1, P5, P10, P90, P95, P99,
etc.
LABEL and FORMAT Statements

**LABEL** variable = ‘label’ … ;

e.g., \[\text{LABEL Pts='Aerobic Points'}\]

\[\text{Percent='\% OF TOTAL FAT'}\]

**FORMAT** variables [format] … ;

e.g., \[\text{FORMAT Weight Height 8.2 Income dollar12.2} \]

- These 2 statements can be used in both DATA and PROC steps
- In a DATA step, these statements will associate **labels** and **formats** permanently with the variable specified
- Also look-up
  - LENGTH
  - INFORMAT
  - ATTRIB

statements
Some Special Pointer Controls

- \#n : for reading multiple data lines when inputting an observation
  - INPUT A 5. B 6.2 \#2 @15 C 7.3 ;
  - INPUT Age Grade/Score1-Score5 ;

- Trailing @ : for holding a data line for another INPUT statement in the same data step
  - DATA ;
    - INPUT Type $1 @ ;
    - IF Type=‘C’ THEN
      - INPUT … ;
    - ELSE
      - INPUT … ;

- Trailing @@ : for holding a data line for repeated execution of the same data step
  - DATA ;
    - INPUT Name $ Verbal Math @@ ;
    - Total=Verbal + Math;
    - DATALINES ;
    - Sue 610 560 John 720 640 Mary 580 590
    - Jim 650 760 Bernard 690 670 Gary 570 680
    - ….
You can write observations into a SAS dataset whenever in a SAS data step instead of at the end of the data step.

data convert;
  do Celsius= -10 to 40 by 5;
    Fahrenheit=9*Celsius/5+32;
    output;
  end;
run;

proc print data=convert;
title "Celsius to Fahrenheit Converter";
run;
OUTPUT Statement

- **OUTPUT** [datasetname] ;

- **Example**

  ```
  data fall09;
  input Name $ Test1-Test3 ;
  drop Test1-Test3 ;
  Test=1; Score=Test1; output;
  Test=2; Score=Test2; output;
  Test=3; Score=Test3; output;
  datalines;
  Smith 67 92 74
  Jones 76 81 72
  ;
  ```

  This data step creates the dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>Test</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>1</td>
<td>67</td>
</tr>
<tr>
<td>Smith</td>
<td>2</td>
<td>92</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>74</td>
</tr>
<tr>
<td>Jones</td>
<td>1</td>
<td>76</td>
</tr>
<tr>
<td>Jones</td>
<td>2</td>
<td>81</td>
</tr>
<tr>
<td>Jones</td>
<td>3</td>
<td>72</td>
</tr>
</tbody>
</table>

- **Example**

  ```
  data yr91 yr92 ;
  input Year Product $ Quantity Sales ;
  if Year=91  then output yr91 ;
  else output yr92 ;
  datalines;
      data lines here
  ;
  ```
Arrays

A sequence of variables may be processed together by first defining them in an array

Example 1:

data scores;
  input Quiz1-Quiz5 Test1-Test3;
  array Scores (8) Quiz1-Quiz5 Test1-Test3;
  do I= 1 TO 8 ;
    if Scores(I)=. then Scores(I)=0 ;
  end ;
datalines;
    ...... ...... ...... ;

Can use with SAS Expressions and can be used in assignment statements and with

DO WHILE, DO UNTIL, IF-THEN, INPUT, PUT, ETC.
Example 2:

data compete (drop=Num);
input Red Blue Grey Green White;
array tms98(5) Red Blue Grey Green White;
Total=0;

do Num=1 to 5;
    if tms98(Num)=. then tms98(Num)=0;
    tms98(Num) = tms98(Num)**2;
    Total + tms98(Num);
end;

datalines;
4  6  0  1  .
3  2  8  9  12
5  .  4  7  6
7  5  10  4  5
;
proc print;
run;
Arrays (Continued)

Arrays could be used in nested DO structures and multidimensional arrays.

See ARRAY statement in SAS Language Reference

Example 3:

data weekly;
input Hours1-Hours7;
array work (7) Hours1-Hours7 ;
array pay (7) Pay1-Pay7 ;
do I= 1 to 7 ;
  if work(I)=999 then work(I)=. ;
  pay(I)= work(I)*12.50 ;
end ;
Example 4:

data fall12;
input Name $ Test1-Test10;
array Test (*) Test1-Test10;
drop Test1-Test10;
do TestNo=1 TO 10;
   Score=Test (TestNo); output;
end;
datalines;
Smith  8   7   9   .   3   10   9   7   5   7
Jones  4   5   6   8   4   5   .   8   7   6
;

Creates the following dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>TestNo</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Smith</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>