TOPICS:
- DATA step
- SET statement
- Creating variables
- IF/THEN IF/THEN ELSE statements
- Functions
- Dates
- Label and Length statements

SAS TABLE AS INPUT
- SAS data set (table) is the result of data definition (DATA step, IMPORT WIZARD)
- SAS data set contains the variables and data for each observation
- SAS may use this file without returning to data definition in DATA and PROC steps
- permanent or temporary

Reading a SAS Data Set
- To create a SAS data set using a SAS data set as input, you must use a DATA statement to start a DATA step and name the SAS data set being created (output data set: newclass2) SET statement to identify the SAS data set being read (input data set: myib.class2).
- To create a variable, you must use an assignment statement to use the values of the variables Weight and Height and assign the result of the calculation to the variable BMI.

Assignment Statements
- An assignment statement evaluates an expression assigns the resulting value to a variable.

Reading a SAS Data Set
General form of a DATA step:
```
DATA output-SAS-data-set;
SET input-SAS-data-set;
additional SAS statements
RUN;
```
- By default, the SET statement reads all of the observations from the input SAS data set variables from the input SAS data set.
Define the Variable

\[
\text{new\_variable\_name} = \text{expression} = \text{EVALUATE}\text{ASSIGN}
\]

\[
\text{BMI} = (\text{weight} \times 0.454) / ((\text{height} \times 0.0254)^2);
\]

Name the New Variable

Rules for naming SAS variables:
- 1 to 32 characters in length
- start with a letter (A through Z) or an underscore (_)
- continue with any combination of numbers, letters, or underscores
- can be stored in mixed-case.

SAS Expressions

An expression contains operands and operators that form a set of instructions that produce a value.

Operands are:
- variable names
- constants.

Operators are:
- symbols that request arithmetic calculations
- SAS functions.

EXAMPLES

- \( x = 3; \) assigns 3 to \( X \) for all observations
- \( y = \text{age}/10; \) assigns the value of age divided by 10 to each observation
- \( \text{Clinic} = \text{’Boston’}; \) assigns the character constant Boston to the variable clinic for each observation
- \( \text{bmi} = \text{wgtkg}/(\text{htm}^2); \) assigns results of calculation to new variable bmi

NOTE: \( X, Y \) and bmi are numeric variables; clinic is a character variable.

Using Operators

Selected operators for basic arithmetic calculations in an assignment statement:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Action</th>
<th>Example</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>( \text{Sum} = \text{x} + \text{y}; )</td>
<td>III</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>( \text{Diff} = \text{x} - \text{y}; )</td>
<td>III</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>( \text{Mult} = \text{x} \times \text{y}; )</td>
<td>II</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>( \text{Divide} = \text{x} / \text{y}; )</td>
<td>II</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>( \text{Raise} = \text{x}^{\text{y}}; )</td>
<td>I</td>
</tr>
<tr>
<td>-</td>
<td>Negative prefix</td>
<td>( \text{Negative} = \text{-x}; )</td>
<td>I</td>
</tr>
</tbody>
</table>

CREATING VARIABLES
RULES FOR NEW VARIABLES

• SAS creates the new variable for all observations in a dataset
• assigns an initial value of . (missing) for numeric data or blank for character data
• follows the standard mathematical rules use parentheses to override the order of calculation

RULES FOR NEW VARIABLES (continued)

• a variable will be assigned a new value only if program statement conditions are satisfied
• if program statement conditions are not met, the value of the variable will remain missing for that observation

RULES FOR NEW VARIABLES (continued)

• variable has the same type and length as the result of the expression on the right side of the statement (character or numeric)
• clinic='Boston'; creates character variable of length 6
• bmi=wgtkg/(htm**2); creates numeric variable

MANIPULATION OF DATA

• requires a DATA step
• reading an existing SAS data with a SET statement
• assignment statements
• functions
• conditional logic

EXAMPLE: Class2_1.SAS

• Survey of home gardeners
• Data file: garden
• Variables:
  name - name of gardener
tomato, zucchini, peas, grapes - number of pounds produced
EXAMPLE: CLASS2_1.SAS

* Modify garden data set with assignment statements;
DATA homegarden; set mylib.garden;
  Zone = 14;
  Type = 'home';
  Zucchini = Zucchini * 10;

PROC PRINT DATA = homegarden;
TITLE 'Home Gardening Survey';
RUN;

RESULTS

- numeric variable Zone created equal to 14
- character variable Type created equal to character constant 'home'
- variable Zucchini is multiplied by 10 and replace with new value
- numeric variable Total created and equal to the sum of all the vegetables

RESULTS

- numeric variable PerTom created equal to the percentage of harvest that are tomatoes
- Total and PerTom variable are missing for the last observation because the variable Pea is missing.
- LOG message indicates missing values were generated

Using SAS Functions

A SAS function is a routine that returns a value that is determined from specified arguments. Must be used in an assignment statement.

General form of a SAS function:

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

Example

```sas
Agegrp=INT(baseage/10);
```

Using SAS Functions

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

Sample statistics functions ignore missing values.
SAS FUNCTIONS
- variable type (character, numeric) depends upon the function
- returns a value -- requires that it be used in some form of assignment.
- calculate sample statistics, create SAS data values, round numbers, extract a portion of a character
- Check HELP menu for syntax

HELP function categories

Numeric Functions
- **MIN** returns smallest value
- **MAX** returns the largest value
- **ROUND** rounds the value to the nearest unit
- **MEAN** computes the arithmetic mean
- **SUM** calculates the sum of the arguments

Using the SUM Function
SUM Sum function returns the sum of the nonmissing arguments (use in an assignment statement)

\[ \text{SUM}(\text{argument1}, \text{argument2}, \ldots) \]

Example:
\[ \text{TOTAL} = \text{SUM}(\text{var}_a, \text{var}_b, \text{var}_c); \]

Using the MEAN Function
MEAN function returns the average of the nonmissing arguments

\[ \text{MEAN}(\text{argument1}, \text{argument2}, \ldots) \]

Example:
\[ \text{Meanvalue} = \text{MEAN}(x_1, x_2, x_3); \]

Using the SUM Function
Selected Variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>total_f</th>
<th>pertom_f</th>
<th>Total</th>
<th>PerTom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gregor</td>
<td>70</td>
<td>14.2857</td>
<td>70</td>
<td>14.2857</td>
</tr>
<tr>
<td>2</td>
<td>Molly</td>
<td>1075</td>
<td>1.3953</td>
<td>1075</td>
<td>1.3953</td>
</tr>
<tr>
<td>3</td>
<td>Luther</td>
<td>215</td>
<td>23.2558</td>
<td>215</td>
<td>23.2558</td>
</tr>
<tr>
<td>4</td>
<td>Susan</td>
<td>40</td>
<td>50.0000</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
Using the INT Function

**INT** Integer function returns the integer portion of the argument (truncates the decimal portion).

**Example:**

```
intvalue = INT(y1/10);
```

---

**EXAMPLE: CLASS2_1.SAS**

Add in these statements by removing the *:

```
mean_prod = mean(tomato, zucchini, peas, grapes);
intvalue = int(mean_prod);
```

---

Using the MEAN and INT Functions

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Total</th>
<th>mean_prod</th>
<th>intvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gregor</td>
<td>70</td>
<td>17.500</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Molly</td>
<td>1075</td>
<td>268.750</td>
<td>268</td>
</tr>
<tr>
<td>3</td>
<td>Luther</td>
<td>215</td>
<td>53.750</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>Susan</td>
<td>.</td>
<td>13.333</td>
<td>13</td>
</tr>
</tbody>
</table>

---

**HELP Menu**

String functions

---

**The SUBSTR Function (Right Side)**

- The SUBSTR function is used to extract or replace characters.

```
NewVar = SUBSTR(string, start, length);
```

- This form of the SUBSTR function (right side of assignment statement) extracts characters.
The SUBSTR Function (Right Side)
- Extract two characters from Location and start at position 11.

State=substr(Location,11,2);

EXAMPLE: Substr Function
- name=“John Doe”;
  first=substr(name,1,4);
  second=substr(name,6,3);
  RESULT: first=“John” and last=“Doe”
  NOTE: length of first and last will be 8 (need length statement to correct)

Another string function:
- name=upcase(name);
  RESULT: name=“JOHN DOE”;

The SCAN Function
- The SCAN function returns the nth word of a character value, each word is separated by delimiters.
- It is used to extract words from a character value when the relative order of words is known, but their starting positions are not.

NewVar=SCAN(string,n,delimiters);
The SCAN Function

- Extract the second word of Phrase.

```plaintext
Second=scan(Phrase,2,':');
```

**EXAMPLE:** Scan Function

- name="John Doe";
  first=scan(name,1);
  second=scan(name,2);

RESULT: first="John" and last="Doe"

NOTE: length of first and last will be 200 (need length statement to correct)

SCAN FUNCTION

Examples (in data step):

```plaintext
arg='ABC.DEF(X=Y)';
word=scan(arg,3);
RESULT: word='X=Y'

word=scan(arg,1,"(");
RESULT: word='ABC.DEF';
* Uses only ( as the delimiter
**LENGTH STATEMENT**
- When creating new variables SAS allocates as many bytes of storage as the number of characters in the first value.
- Specify the length with this statement:

```
LENGTH variable1..variable n ($) length;
```

$ indicates preceding vars are character
length is a numeric constant -- # of bytes.

Example:
```
Length finit limit $ 1;
```

---

**Character Functions**
- **SCAN** separates a string into words and returns a specific word.
- **SUBSTR** extracts a substr (portion of a character string).
- **TRIM** trims trailing blanks.
- **INDEX** searches for a specific string and returns the position of that string.

---

**Children SAS dataset: Selected Records**

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred Koenig</td>
<td>M</td>
<td>12</td>
<td>69</td>
<td>112.5</td>
</tr>
<tr>
<td>Alice Lindermann</td>
<td>F</td>
<td>11</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara Lichtermann</td>
<td>F</td>
<td>11</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>John P. Wallace</td>
<td>M</td>
<td>10</td>
<td>59</td>
<td>99.5</td>
</tr>
<tr>
<td>Joyce K. Kumar</td>
<td>F</td>
<td>9</td>
<td>51.3</td>
<td>50.5</td>
</tr>
</tbody>
</table>

---

**Example : CLASS2_2.SAS**

- Creates 3 new variables from the variable name using the SCAN function:
  - First is first word in name
  - Second is the second word
  - Third is the third word

Separate variables for each part of the name variable.

---

**Example : CLASS2_2.SAS**

- `agegp` variable created using INT function and the age variable.
- Values of weight and height are replaced with metric.
- Numeric variable `bmi` created using new weight and height values.
- `Finit` variable created – first initial using SUBSTR function.

---

**Example : CLASS2_2.SAS**

Data child; set mylib.children;
Length first second $8 third $25;
height=height*0.0254;
* convert to meters;
weight=weight*.454;
* convert to kilograms;
bmi=weight/(height**2);
Example: CLASS2_2.SAS

```sas
first=scan(name,1,"";
second=scan(name,2,"";
third=scan(name,3,"";
agegp=int(age/10);
finit=substr(first,1,1);
```

Example: CLASS2_2.SAS

- Look at the length of the variable second and finit

How do we fix the program?

EXERCISE I

CONDITIONAL LOGIC

Objectives

- Execute statements conditionally using IF-THEN logic.
- Control the length of character variables explicitly with the LENGTH statement.
- Select rows to include in a SAS data set.
- Use SAS date constants.

Conditional Execution

- General form of IF-THEN and ELSE statements:

```
IF expression THEN statement;
ELSE statement;
```

Expression contains operands and operators that form a set of instructions that produce a value.

Operands are
- variable names
- constants.

Operators are
- symbols that request
  - a comparison
  - a logical operation
  - an arithmetic calculation
- SAS functions.

Only one executable statement is allowed on an IF-THEN or ELSE statement.
### Comparison and Logical Operators

- **EQ** or **=** = equal to
- **NE** or **^=** = not equal to
- **LT** or **<** = less than
- **GT** or **>** = greater than
- **LE** or **<=** = less than or equal to
- **GE** or **>=** = greater than or equal to
- **&** = and
- **|** = or
- **^** or **~** = not

### Conditional Execution

Assign value of last based on the value of the variable THIRD. Add these statements to the CLASS2_2.SAS file by removing the *

```sas
if third='' then last=second;
else last=third; run;
```

---

### Child SAS dataset: Selected Records

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>Koenig</td>
<td>Koenig</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>Lindermann</td>
<td>Lindermann</td>
<td></td>
</tr>
<tr>
<td>Barbara</td>
<td>Lichtermann</td>
<td>Lichterman</td>
<td></td>
</tr>
<tr>
<td>Philip J.</td>
<td>Kirk</td>
<td>Kirk</td>
<td></td>
</tr>
<tr>
<td>Robert H.</td>
<td>Goodnight</td>
<td>Goodnight</td>
<td></td>
</tr>
</tbody>
</table>

---

### Conditional Execution

- Create a new variable BMIGRP based on the value of BMI

Newclass2 SAS dataset created with Class2_3.sas

<table>
<thead>
<tr>
<th>Name</th>
<th>Birthdate</th>
<th>Basage</th>
<th>Height</th>
<th>Height</th>
<th>BMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richardson</td>
<td>2225</td>
<td>14</td>
<td>149</td>
<td>72</td>
<td>25.6558</td>
</tr>
<tr>
<td>Lowrey</td>
<td>3734</td>
<td>29</td>
<td>235</td>
<td>71</td>
<td>32.8050</td>
</tr>
<tr>
<td>Tierney</td>
<td>52</td>
<td>229</td>
<td>70</td>
<td>32.8873</td>
<td></td>
</tr>
<tr>
<td>Sommers</td>
<td>-205</td>
<td>23</td>
<td>156</td>
<td>66</td>
<td>25.2014</td>
</tr>
<tr>
<td>Kegan</td>
<td>7525</td>
<td>19</td>
<td>140</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Create bmigrp variable based on value of bmi

```plaintext
IF bmi<26 THEN bmigrp=1;
ELSE IF bmi>=26 THEN bmigrp=2;
run;
```

PDV (First Observation)

<table>
<thead>
<tr>
<th>Lname</th>
<th>Birthdate</th>
<th>Baseage</th>
<th>Weight</th>
<th>Height</th>
<th>Bmi</th>
<th>bmigrp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richardson</td>
<td>2225</td>
<td>34</td>
<td>189</td>
<td>72</td>
<td>25.65</td>
<td>1</td>
</tr>
</tbody>
</table>

PDV (Second Observation)

<table>
<thead>
<tr>
<th>Lname</th>
<th>Birthdate</th>
<th>Baseage</th>
<th>Weight</th>
<th>Height</th>
<th>Bmi</th>
<th>bmigrp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowery</td>
<td>3724</td>
<td>29</td>
<td>235</td>
<td>71</td>
<td>32.8</td>
<td>2</td>
</tr>
</tbody>
</table>

PDV (Fifth Observation)

<table>
<thead>
<tr>
<th>Lname</th>
<th>Birthdate</th>
<th>Baseage</th>
<th>Weight</th>
<th>Height</th>
<th>Bmi</th>
<th>bmigrp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kegan</td>
<td>7525</td>
<td>19</td>
<td>140</td>
<td>71</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Evaluating Missing Values

- Do you want observation 5 to bmigrp 1?
- Why does this happen?
  Period or a blank value is less than zero
- How do you fix this?
  Take into account possible missing values in the IF/THEN statements

![Conditional Execution](image)

**Conditional Execution**

```sas
IF bmi= THEN; *NO ACTION TAKEN; 
ELSE IF bmi<26 THEN bmigrp=1; 
ELSE IF bmi>=26 THEN bmigrp=2; 
* Alternative last statement 
ELSE bmigrp=2; run;
```

**PDV (Fifth Observation)**

<table>
<thead>
<tr>
<th>Lname</th>
<th>Birthdate</th>
<th>Baseage</th>
<th>Weight</th>
<th>Height</th>
<th>BMI</th>
<th>bmigrp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kegan</td>
<td>7525</td>
<td>19</td>
<td>140</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

Conditional Execution: ALTERNATE

```sas
IF 0<bmi<26 THEN bmigrp=1; 
ELSE IF bmi>=26 THEN bmigrp=2; 
*No action is taken and bmigrp remains missing; 
run;
```

**PDV (Fifth Observation)**

<table>
<thead>
<tr>
<th>Lname</th>
<th>Birthdate</th>
<th>Baseage</th>
<th>Weight</th>
<th>Height</th>
<th>BMI</th>
<th>bmigrp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kegan</td>
<td>7525</td>
<td>19</td>
<td>140</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

Conditional Execution

- You can use the DO and END statements to execute a group of statements based on a condition.

```sas
IF expression THEN DO: 
  executable statements
END; 
ELSE DO: 
  executable statements
END;
```

General form of the DO and END statements:

Comparison and Logical Operators

- EQ or = equal to
- NE or ^= not equal to
- LT or < less than
- GT or > greater than
- LE or <= less than or equal to
- GE or >= greater than or equal to
  & and
  | or
  ^ or ~ not
IF/THEN DO STATEMENTS

IF <condition> THEN DO;
   <action>;
   <action>;
   etc.
END;

where:
condition is a expression comparing one thing to another

actions are the result if the condition is true; otherwise there is no action

DO STATEMENT

- the simplest form of DO group processing.
- statements between the DO and END statements are called a DO group
- can nest DO statements within DO groups
- simple DO statement is often used within IF-THEN/ELSE statements to designate a group of statements to be executed depending on whether the IF condition is true or false.

EXAMPLE: IF/THEN DO (CLASS2_2)

if third="" then do;
   last = second;
   second="";
   end;
else last =third;

All statements after the DO until the END statement are executed for those observations where the condition is met (third="")

Child SAS dataset: Selected Records

<table>
<thead>
<tr>
<th>first</th>
<th>second</th>
<th>Third</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td></td>
<td>Koenig</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td></td>
<td>Lindermann</td>
<td></td>
</tr>
<tr>
<td>Barbara</td>
<td></td>
<td>Lichterman</td>
<td></td>
</tr>
<tr>
<td>Philip J.</td>
<td></td>
<td>Kirk</td>
<td>Goodnight</td>
</tr>
<tr>
<td>Robert H.</td>
<td></td>
<td>Goodnight</td>
<td>Goodnight</td>
</tr>
</tbody>
</table>

MULTIPLE CONDITIONS

- specify multiple conditions using keywords AND and OR

Syntax:

IF <condition> AND <condition> THEN <action>;

IF <condition> OR <condition> THEN <action>;

EXAMPLE: MULTIPLE CONDITIONS

- AND requires both conditions to be true for the result to be true

IF RACE=1 AND SEX=1 THEN GROUP=1;

- OR allows either condition to be true

IF CLINIC=1 OR CLINIC=2 THEN REGION=1;
ALTERNATIVE ACTION

IF THEN/ ELSE STATEMENTS
- provide alternative action when condition is false
- replaces a series of ifs
- use multiple Else statements to specify a series of mutually exclusive conditions

IF THEN/ ELSE STATEMENTS
- create a categorical variable from a continuous variable
- add the keyword ELSE to IF/THEN statement - tells SAS the statements are related

Example
Series of IF statements:
If sbp>0 and sbp<110 then grp=1;
If sbp>=110 and sbp<=130 then grp=2;
If sbp>130 then grp=3;
If sbp=999 then grp=.;

Example
Series of IF THEN/ELSE statements:
If sbp=999 then grp=.;
Else if sbp<110 then grp=1;
Else if <=130 then grp=2;
Else grp=3;

- ELSE should be used for mutually exclusive categories since the use of ELSE prevents the evaluation of an observation by subsequent statements if the conditions in the current statement have been met.
EXAMPLE

IF bmi=. THEN ;
ELSE IF bmi<=24 THEN bmigrp=1;
ELSE IF bmi<=27 THEN bmigrp=2;
ELSE IF bmi>27 THEN bmigrp=3;

* Alternative last statement
ELSE bmigrp=3;

EXAMPLE: IF/THEN DO

IF cvd=1 THEN DO;
cvdtime=cvdyr - entry;
cvdage=cvdyr - yob;
END;
ELSE DO;
cvdtime=follyr - entry;
cvdage = follyr - yob;
END;

IF/THEN DO STATEMENTS

IF <condition> THEN DO;
<action> ; <action> ;
etc.
END;
ELSE IF <condition> THEN DO;
<action> ; <action> ; END;
etc...
ELSE DO;
<action> ; <action> ; END;

EXERCISE II

WORKING WITH SAS DATES

SAS DATES

- dates written with numbers, letters, and characters stored in numeric format
- stored a SAS date value - the number of days since 1/1/60; dates before 1/1/60 stored as negative
- requires formatted input
Dates
SAS date values (numeric values).

<table>
<thead>
<tr>
<th>Store (01JAN1960)</th>
<th>01JAN1961</th>
<th>14DEC2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display (01/01/1960)</td>
<td>01/01/1961</td>
<td>12/14/2000</td>
</tr>
</tbody>
</table>

SAS REPRESENTATION OF DATE CONSTANTS

'DDMMYY'd
where dd is a 2 digit day (02)
mon is a 3 character month (nov)
yy is 2 digit year (99)
Example:
Date1='02nov99'D;

Using SAS Date Constants
The constant 'ddMMmyyy'd (example: '01jan1970'd) creates a SAS date value from the date enclosed in quotes.

- **dd** is a one- or two-digit value for the day.
- **MMM** is a three-letter abbreviation for the month (JAN, FEB, MAR, and so on).
- **yyyy** is a two- or four-digit value for the year.
- **d** is required to convert the quoted string to a SAS date.

**Printing Dates**

```sas
PROC PRINT data=example;
Var id birthdate;
Format mmddyy10.;  SPECIFY FORMAT
Run;
Obs id      birthdate
1     14632    03/04/1993
2     67456    05/28/1991
Without format
1     14632      12116
2     67456      11470
```

**SAMPLE SAS DATA FORMATS**

Print: Sas format:
040197 MMDDYY6.
04/01/97 MMDDYY8.
04/01/1997 MMDDYY10.
010497 DDMMYY6.
01/04/97 DDMMYY8.
01/04/1997 DDMMYY10.
01APR97 DATE7.
01APR1997 DATE9.

**TO RECODE A SAS DATE**

Date1 must be previously defined as a SAS date

```sas
IF date1= . THEN
date1='01APR97'D;
```
or

```sas
IF date1 = . THEN date1=13605;
```
(number of days since 1/1/60)
Using SAS Date Constants

```sas
if birthdate='10jun1959'd then
  birthdate='10sep1959'd;
```

Using Date Functions

- You can use SAS date functions to create SAS date values extract information from SAS date values.

Date Functions: Create SAS Dates

- TODAY() obtains the date value from the system clock.
- MDY(month, day, year) uses numeric month, day, and year values to return the corresponding SAS date value.

Date Functions: Extracting Information

- YEAR(SAS-date) extracts the year from a SAS date and returns a four-digit value for year.
- QTR(SAS-date) extracts the quarter from a SAS date and returns a number from 1 to 4.
- MONTH(SAS-date) extracts the month from a SAS date and returns a number from 1 to 12.
- WEEKDAY(SAS-date) extracts the day of the week from a SAS date and returns a number from 1 to 7, where 1 represents Sunday, and so on.

Using the TODAY Function

- Add an assignment statement to the DATA step to create a variable is the age as of today. (CLASS2_4.SAS)

```sas
age2005 = (today() - birthdate) / 365.25;
```

Calculating Age in 2005

```sas
proc print data=newclass2;
  var baseage agegrp age2005 birthdate;
  format birthdate mmddyy10.;
run;
```

<table>
<thead>
<tr>
<th>The SAS System</th>
<th>Obs</th>
<th>baseage</th>
<th>agegrp</th>
<th>age2005</th>
<th>birthdate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>34</td>
<td>3</td>
<td>39</td>
<td>02/03/1966</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29</td>
<td>2</td>
<td>35</td>
<td>03/23/1970</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>32</td>
<td>3</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>33</td>
<td>3</td>
<td>45</td>
<td>06/10/1959</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>19</td>
<td>1</td>
<td>24</td>
<td>08/08/1980</td>
</tr>
</tbody>
</table>
Using the WEEKDAY Function

• Add an assignment statement to the DATA step to create a variable that shows the day of the week of the date of birth. (CLASS2_4.SAS)

```
birth_dayweek=weekday(birthdate);
```

Calculating Birth Day of Week

```
proc print data=newclass2;
var baseage agegrp birth_weekday birthdate;
format birthdate weekdate.;
run;
```

The SAS System

<table>
<thead>
<tr>
<th>baseline</th>
<th>agegrp</th>
<th>birth_weekday</th>
<th>birthdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>5</td>
<td>5</td>
<td>Thursday, February 3, 1966</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>2</td>
<td>Monday, March 25, 1970</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>4</td>
<td>Wednesday, June 10, 1959</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>6</td>
<td>Friday, August 8, 1980</td>
</tr>
</tbody>
</table>

What if you do not want the variables Height and Weight in the data set?

EXAMPLE: Class2_4.SAS

The following code has been added to the Class2_3.sas program

```
agegrp=int(baseage/10);
if birthdate='10jun1959'd then
  birthdate='10sep1959'd;
age2005 =
int((today()-birthdate)/365.25);
birth_weekday=weekday(birthdate);```

RESULTS

• reads in permanent file class2 and creates temporary file newclass2
• computes new variable agegrp from baseage
• Recodes birthdate from 6/10/1959 to 9/10/1959
• creates new variables age2005 using INT and TODAY functions
• Creates birth_weekday using weekday function

LABEL STATEMENT

• assigns descriptive labels (up to 40 characters) to variables.
• default – SAS uses the variable names

```
LABEL bmi='body mass index'
bmigrp='0=bmi<24 1=bmi>=24';
```

MORE DATA STEP STATEMENTS
DROP AND KEEP STATEMENTS

- **DROP** - specifies the names of the variables you want to omit from the output data set. (subsets the variables)

  \[
  \text{DROP } \text{variable-list} ;
  \]

  Example: \( \text{DROP wgt hgt;} \)

- **KEEP** - specifies the names of the variables to write to the output data set.

  \[
  \text{KEEP } \text{variable-list} ;
  \]

  Example: \( \text{KEEP ID bmi bmigrp;} \)

Example: Class2_5.SAS

- Insert the following statements into the Data step in class2_4.sas

  \[
  \text{label bmi='body mass index'}
  \]

  \[
  \text{bmigrp='1= <26 2= >=26';}
  \]

  \[
  \text{drop lname height weight;}
  \]

Exercise III-IV

Program Checking

- run on a subset of the data
- use a OBS option on SET statement to limit amount of processing
- have a “test” data
- check recodes by printing out a few records and checking all calculations and recodes
- check the number of missing
- use procedures such as FREQ MEANS

Look at the data!

Look at the program!

Look at the log!

Look at the output!
● Topics
  DATA STEP statements
  SET statement
  Data set options
  Merging files

● References
  The Little SAS Book - Chapter 5