Loops & Arrays

efficiency
for statements
while statements

Hye-Chung Kum
Population Informatics Research Group
http://pinformatics.org/

License:
Data Science in the Health Domain by Hye-Chung Kum is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License

Course URL:
http://pinformatics.org/phpm672

1

What you learned so far…

- Assignment 1
  - Setup work environment
  - Use the SAS software
  - SAS programming basics
    - data step & proc step
    - libname
    - Writing code & Reading logs
- Assignment 2
  - Understand variables (names, types, labels)
  - To write conditional logic codes
  - Subset columns (variables) from a table
  - Subset rows (observations) from a table
  - Recode, rename variables and calculate new variables
  - Label variables and values

2

Assignment Plan

- 1: Type what I gave you and run
- 2: Write your own relatively simple
- 3: Write your first real program (reusable elegant code)
- 4: Combining Tables
- 5: Indexing
- 6: Macros
- Final project

POPULATION INFORMATICS

3

Required Reading

- UCLA module
  - https://stats.idre.ucla.edu/sas/modules/working-across-variables/
- Little SAS book
  - 3.11 Simplifying programs with arrays
  - 3.12 Using Shortcuts to Lists of Variable Names
- Most difficult of required content
  - assignment 1 to 4
- But also will come in most handy in doing your research
- READ the required readings
- Attend Lab tomorrow

POPULATION INFORMATICS

4

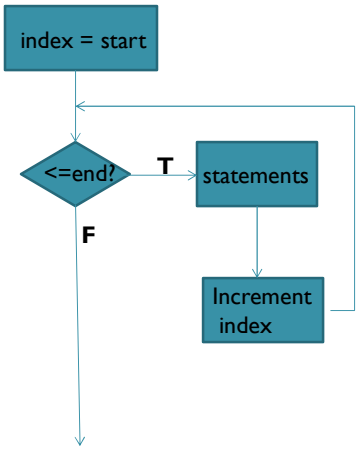
Objective

- use for loops (counting loops)
- use while loops (conditional loops)
- use one dimensional arrays
- Understand how to write reusable code
- Understand how to optimize your programming time: KISS (Keep it simple)

POPULATION INFORMATICS

5

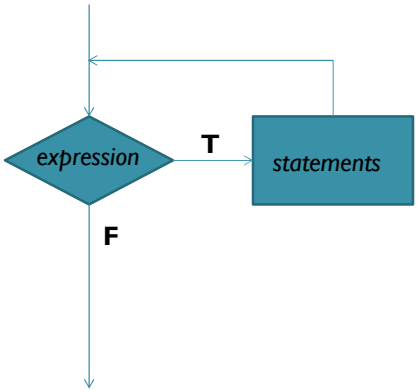
do *index = start to end by increment;*
statements;
end;



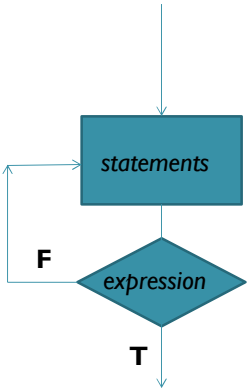
POPULATION INFORMATICS

6

do while (*expression*);
statements;
end;



do until (*expression*);
statements;
end;



POPULATION INFORMATICS

7

Programming Goals:

- **Correctness**
  - Gives the right answer
  - Never returns the wrong answer
- **Robustness**
  - Program doesn't crash, even for bad input
- **Maintainable (or *Sustainable*)**
  - Simple code, easy to understand and modify
  - Readable, well-commented, well-structured
- **Fast (Efficient)**
  - Uses efficient algorithms
  - Takes advantage of language features to improve speed

POPULATION INFORMATICS

8

User Efficiency

optimize your own time

- K.I.S.S. Keep it simple …
  - Simple code is easier to understand and fix
  - A simple but **correct** solution is more valuable than a clever elegant but **incorrect** solution.
- Understand your code, **Avoid accidental coding**
  - Find some code, type it in, it seems to work, so …
  - When problems inevitably appear, you can't fix the bugs, if you don't understand your own code…
  - Use help & documentation
  - Play with functionality until you understand it. (trial & error)
- Have a plan (Divide & Conquer)
  - Come up with a plan
  - Break plan into small bite-size chunks
  - Solve each chunk and verify that chunk works properly
  - Assemble all the working chunks to solve original problem

POPULATION INFORMATICS

9

Algorithmic Efficiency

- Reducing the amount of computing resources that an algorithm consumes
  - **Speed:** The amount of time it takes for an algorithm to complete
  - **Space:** The amount of memory or storage used by an algorithm.
- ***Note:*** Most of the problems we solve in class don't require this extra level of effort.
- If your solution works correctly, but is running too slowly, or is taking too much memory, often the best solution is to find a better algorithm.

POPULATION INFORMATICS

10

Looping Efficiency

- **Loops** are powerful flexible concepts for solving problems involving repetitive processing of the same task with different data over and over again
- It makes modifying code efficient
  - You don't have to change in multiple places

POPULATION INFORMATICS

11

Looping

Goal: I have a task (piece of code) that I want to repeat over and over again on a list of data.

How could I do that?

```
* Brute Force:  Cut & Paste & Tweak
if cigever=1 then bcigever=1;
else if cigever=2 then bcigever=0;

if alcever=1 then balcever=1;
else if alcever=2 then balcever=0;

if cocever=1 then bcocever=1;
else if cocever=2 then bcocever=0;

if mjever=1 then bmjever=1;
else if mjever in (0,2) then bmjever=0;
```

POPULATION INFORMATICS

12

Arrays

array{1}	array{2}	array{3}	array{4}
rate2005	rate2006	rate2007	rate2008

- A set of variables grouped together for the duration of the data step
- So that all variables in the group can be referred to systematically
- SAS: index typically starts at 1
- Every task that can be done with arrays can also be done without arrays
- Why do we use arrays?
  - Efficient programming: do not need to write repeated codes
  - Accuracy: With fewer lines of codes, easier to debug ERRORs, and maintain code
  - Extensible: Easy to extend your code

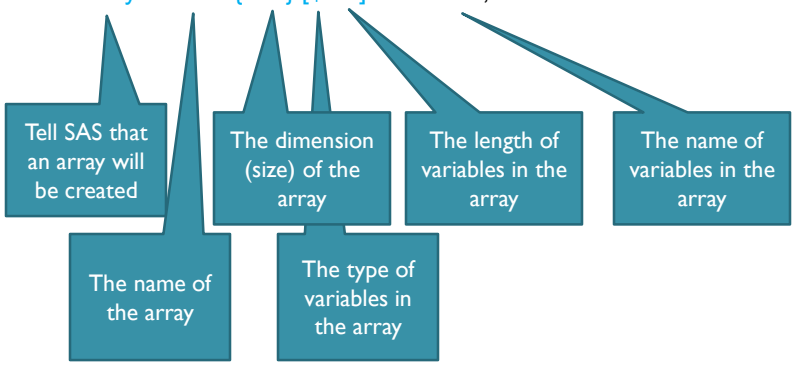
Population Informatics

13

SAS: Arrays

array{1}	array{2}	array{3}	array{4}
rate2005	rate2006	rate2007	rate2008

- array aname {dim} [$len] elements;



- array rate {4} rate2005-rate2008;

Population Informatics

14

SAS: Arrays

array{1}	array{2}	array{3}	array{4}
rate2005	rate2006	rate2007	rate2008

- All variables in one array must be of the same type
- Variables specified within an array do not need to already exist
- array aname {dim} [$len] elements
  - array rate {4} rate2005-rate2008;
  - array rate {*} rate2005-rate2008;
  - array rate {4} ; *implicit: rate1-rate4;
  - array rate {*} rate: ; *NOT RECOMMENDED;
- Dim(Dimension): how many elements
  - Can be implicit by using *
- $len: type and length of variables when strings
  - Omitted for numerical variables
  - Array name{3} $10.;
- elements: list of variables
- index: an integer pointer that identifies the element in the array
  - array {index} or array [index]
  - rate2006 is indexed by 2

POPULATION INFORMATICS

15

Lab 3 Objective

- use for loops (counting loops)
- use while loops (conditional loops)
- use one dimensional arrays

POPULATION INFORMATICS

16

Start Lab 3

- Who does not have lab2 working? Download from site
- Not allowed to open the full table ever for this class, even if you can.
  - Purpose is to learn to use BIG tables
- Option1: having difficulty reading code
  - Submit fully commented code
    - Add line by line comments (i.e. translate into English)
    - Important to understand what each line does
  - Submit log & results
    - Read the log to understand and add comments
- Option 2: comfortable reading code, not writing code
  - Read my code
  - Try to write it starting from lab2, without looking
- Option 3: comfortable reading & writing code
  - Do small exercise: write code (P2)

POPULATION INFORMATICS

17

By next class

- Read lab 3 and assignment 3
- Ask questions
- There is a midpoint submission
- You have 2.5 weeks on this assignment
  - Midpoint at 1.5 weeks (Tues)
- In class
  - Review assignment 3 midpoint email together
  - Website
  - Diff lab2.sas lab3_for.sas

POPULATION INFORMATICS

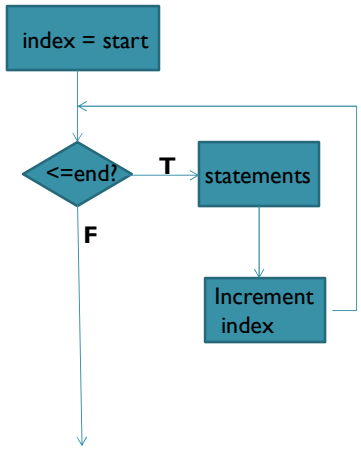
18

Counted (Iterative) Loops

POPULATION INFORMATICS

19

do *index = start to end by increment;*
    statements;
end;



POPULATION INFORMATICS

20

SAS: for loop statement

the **counted loop** solution

```
do <varindex> = <start> to <stop>;
     <Body: do some work with varindex>
end;

do <idx> = <start> to <stop> by <step>;
     <Body: do some work with varindex>
end;
```

POPULATION INFORMATICS

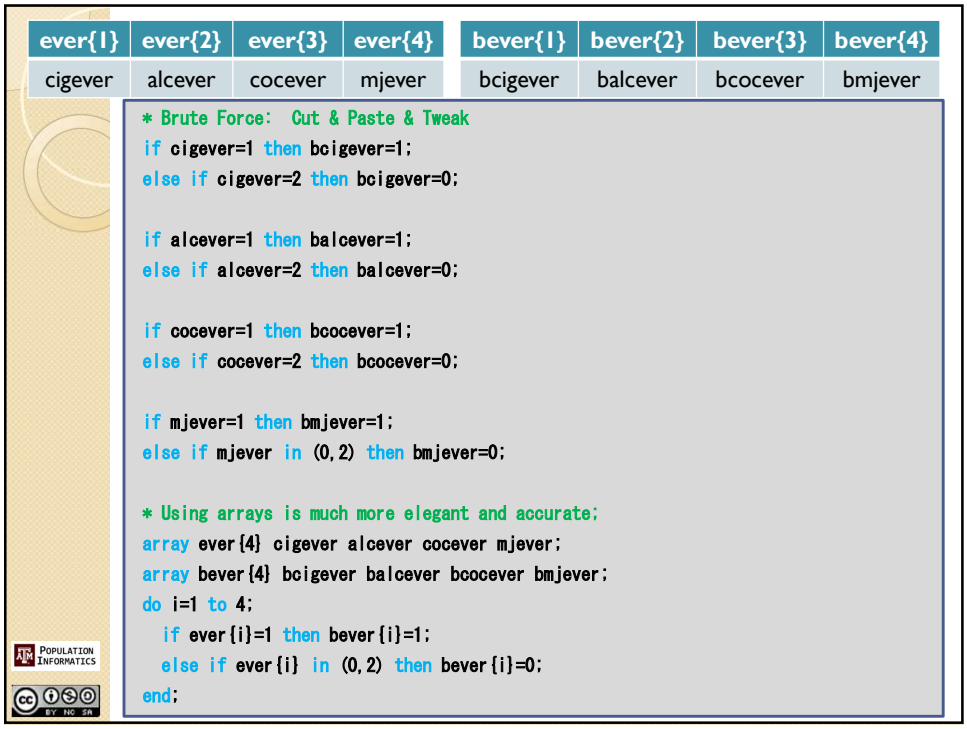
21

ever{1}	ever{2}	ever{3}	ever{4}	bever{1}	bever{2}	bever{3}	bever{4}
cigever	alcever	cocever	mjever	bcigever	balcever	bcocever	bmjever

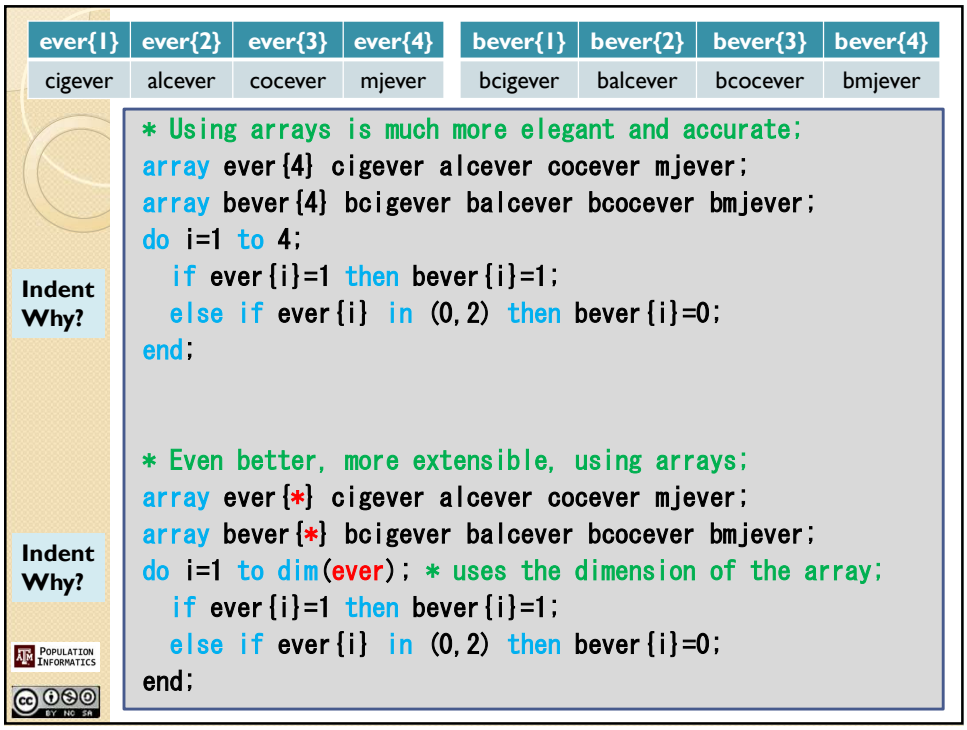
```
* Brute Force:  Cut & Paste & Tweak
if cigever=1 then bcigever=1;
else if cigever=2 then bcigever=0;

if alcever=1 then balcever=1;
else if alcever=2 then balcever=0;

if cocever=1 then bcocever=1;
else if cocever=2 then bcocever=0;

if mjever=1 then bmjever=1;
else if mjever in (0,2) then bmjever=0;

* Using arrays is much more elegant and accurate;
array ever{4} cigever alcever cocever mjever;
array bever{4} bcigever balcever bcocever bmjever;
do i=1 to 4;
  if ever{i}=1 then bever{i}=1;
  else if ever{i} in (0,2) then bever{i}=0;
end;
```

POPULATION INFORMATICS

22


ever{1} ever{2} ever{3} ever{4} bever{1} bever{2} bever{3} bever{4}
cigever alcever cocever mjever bcigever balcever bcocever bmjever
* Brute Force: Cut & Paste & Tweak
if cigever=1 then bcigever=1;
else if cigever=2 then bcigever=0;
if alcever=1 then balcever=1;
else if alcever=2 then balcever=0;
if cocever=1 then bcocever=1;
else if cocever=2 then bcocever=0;
if mjever=1 then bmjever=1;
else if mjever in (0,2) then bmjever=0;
* Using arrays is much more elegant and accurate;
array ever{4} cigever alcever cocever mjever;
array bever{4} bcigever balcever bcocever bmjever;
do i=1 to 4;
if ever{i}=1 then bever{i}=1;
else if ever{i} in (0,2) then bever{i}=0;
end;
POPULATION INFORMATICS

23


ever{1} ever{2} ever{3} ever{4} bever{1} bever{2} bever{3} bever{4}
cigever alcever cocever mjever bcigever balcever bcocever bmjever
Indent Why?
* Using arrays is much more elegant and accurate;
array ever{4} cigever alcever cocever mjever;
array bever{4} bcigever balcever bcocever bmjever;
do i=1 to 4;
if ever{i}=1 then bever{i}=1;
else if ever{i} in (0,2) then bever{i}=0;
end;
Indent Why?
* Even better, more extensible, using arrays;
array ever{*} cigever alcever cocever mjever;
array bever{*} bcigever balcever bcocever bmjever;
do i=1 to dim(ever); * uses the dimension of the array;
if ever{i}=1 then bever{i}=1;
else if ever{i} in (0,2) then bever{i}=0;
end;
POPULATION INFORMATICS

24

ever{1}	ever{2}	ever{3}	ever{4}	bever{1}	bever{2}	bever{3}	bever{4}
cigever	alcever	cocever	mjever	bcigever	balcever	bcocever	bmjever

Indent Why?

```
* Using arrays is much more elegant and accurate;
array ever{5} cigever alcever cocever mjever snfever;
array bever{5} bcigever balcever bcocever bmjever bsnfever;
do i=1 to 5;
  if ever{i}=1 then bever{i}=1;
  else if ever{i} in (0,2) then bever{i}=0;
end;


* Even better, more extensible, using arrays;
array ever{*} cigever alcever cocever mjever snfever;
array bever{*} bcigever balcever bcocever bmjever bsnfever;
do i=1 to dim(ever); * uses the dimension of the array;
  if ever{i}=1 then bever{i}=1;
  else if ever{i} in (0,2) then bever{i}=0;
end;
```

Indent Why?

POPULATION INFORMATICS

25

Indentation – helps outline code

Which is more readable?

```
do i=1 to dim(ever);
  if ever{i}=1 then
    bever{i}=1;
  else if ever{i} in (0,2) then
    bever{i}=0;
end;
```

```
do i=1 to dim(ever);
if ever{i}=1 then
bever{i}=1;
else if ever{i} in (0,2) then
bever{i}=0;
end;
```

POPULATION INFORMATICS

26

Indentation & Line Break
Which is more readable?

```
do i=1 to dim(ever);
  if ever{i}=1 then
    bever{i}=1;
  else if ever{i} in (0,2) then
    bever{i}=0;
end;
```

```
do i=1 to dim(ever);
  if ever{i}=1 then bever{i}=1;
  else if ever{i} in (0,2) then bever{i}=0;
end;
```

POPULATION INFORMATICS

27

Looping behavior (Iteration)

```
do i=1 to dim(ever);
  if ever{i}=1 then bever{i}=1;
  else if ever{i} in (0,2) then bever{i}=0;
end;
```

Body:
This code gets repeated 'n' times,
n = dim(ever) = 4

```
* Hidden Code:   i = i + 1;   * changes each iteration
  Inserted Here  if i <= dim(ever)
                   <jump back to top of loop>
                 else <exit loop> end
```

POPULATION INFORMATICS

28

How to figure out new syntax

- Changes over time
- Find a reliable source you like
- https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=pgmsashome&docsetTarget=home.htm&locale=en
  - Language elements/statements/do
- http://www.stata.com/help.cgi?foreach
- google
  - sas loops
  - sas arrays
  - stata foreach over multiple varlist
  - http://www.stata.com/statalist/archive/2013-03/msg01241.html

POPULATION INFORMATICS

29

Counted Loops

Code some

POPULATION INFORMATICS

30

Counted Loops vs. Conditional Loops

- **Counted Loops**
  - I want to repeat a task (piece of code) a specified number of times, say `'n'`
    - **Example:** I want to calculate grades for all 40 students in my class
- **Conditional Loops**
  - I want to repeat a task until some condition is satisfied.
    - **Example:** I want to grade as many students as I can between now and when I go home at 5:00 PM.

POPULATION INFORMATICS

31

SAS: conditional loops

- There are 3 forms of the DO statement:
  - The iterative DO statement executes statements between DO and END statements repetitively based on the value of an index variable. The iterative DO statement can contain a WHILE or UNTIL clause.
    - STOP when finished running N times
  - The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop.
    - STOP when the condition is TRUE
  - The DO WHILE statement executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop.
    - STOP when the condition is FALSE

POPULATION INFORMATICS

32

do while loop statement

the **conditional loop** solution (SAS)

```
do while (<test>);
    <Body: do some work>
    <Update: make progress towards exiting loop>
end;
```

If we don't know ahead of time, how many times we need to loop but we can write a **test** for when we are done; Then the **while** loop is a great solution.

Note: For this to work properly, the <test> needs to evaluate to a logical value.

Note: The body of the **`while`** loop will continue to get executed as long as the <test> evaluates to `true`. The while loop is exited as soon as the condition evaluates to `false`.

POPULATION INFORMATICS

33

do until loop statement

the **conditional loop** solution

```
do until (<test>);
    <Body: do some work>
    <Update: make progress towards exiting loop>
end;
```

- Very similar to **do while** loop
- The difference ?
  - The **test** is evaluated
    - Until: at the **bottom** of the loop **after** the statements in the DO loop have been executed. **The DO loop always iterates at least once.**
    - While: at the **top** of the loop **before** the statements in the DO loop have been executed.
  - Stops when
    - Until: If the expression is **true**, the DO loop does not iterate again
    - While: If the expression is **false**, the DO loop does not iterate again.

POPULATION INFORMATICS

34

Infinite Loops

```
count = 1;
do while (1);  * test always true;
   * This Loop never stops;
   count = count + 1;
end;
```

Note: Use `<ctrl-c> or STOP or Kill SAS` to exit current execution, if you appear to be stuck in an infinite loop.

For most programs, the **test** expression must eventually become *`false`*, for the loop to be useful.

POPULATION INFORMATICS

35

Counting in a while loop

```
* Initialize variables;
array rate{*} rate2001 - rate2013;
idx = 1;
count = 0;

* Count years with rate > 7;
do while (idx <= dim(rate));

   * Test current element against 7;
   if rate(idx) > 7.0 then
      count = count + 1;

   * Update:  Don't forget to increment !;
   idx = idx + 1;
end;
```

POPULATION INFORMATICS

36

Better to use the for loop

```
* Initialize variables;
array rate{*} rate2001-rate2013;
count = 0;

* Count years with rate > 7;
do idx=1 to dim(rate));
   * Test current element against 7;
   if rate(idx) > 7.0 then
      count = count + 1;
end;
```

POPULATION INFORMATICS

37

A good example for while loop
multiple conditions

```
* What year was the 4th year when rate > 7;
array rate{*} rate2001 - rate2013;
idx = 1;
count = 0;

* Count years with rate > 7;
do while (count<4 & idx <= dim(rate));
   * Test current element against 7;
   if rate(idx) > 7.0 then
      count = count + 1;

   * Update:  Don' t forget to increment !
   idx = idx + 1;
end;

if (count=4) then year4=1999+idx;
* else year4=.;
```

POPULATION INFORMATICS

38

leave statement

Terminates `for` or `while` loops. breaks flow of control of inner most nested `while` or `for` loop containing `leave` statement.

```
array rate{*} rate2001 - rate2013;
idx = 1;
count = 0;

* What year was the 4th year when rate > 7;
do while ( idx <= dim(rate) );
  if rate(idx) > 7.0 then
      count = count + 1;

  * Jump out of while loop;
  if (count = 4) then leave;
  idx = idx + 1;
end;
* Control flow jumps to here after break;
if (count=4) then year4=2000+idx;
```

POPULATION INFORMATICS

39

Breaking out of loop

- The LEAVE statement causes processing of the current loop to end.
- The CONTINUE statement stops the processing of the current iteration of a loop and resumes with the next iteration.

POPULATION INFORMATICS

40

Common Pitfalls

- Forgetting to initialize useful variables
  - Remember to set the running sum or count to zero before you start summing or counting.
  - Remember to set the running product to one before using it
  - Remember to initialize index variables for while loops
- Code not executing
  - Not realizing that it is possible for the body of a while loop to never get executed, depending on your **test** condition.
- Causing an Infinite loop
  - Writing a `while` **test** condition that never fails.
  - Forgetting to **update** index variables in `while` loops

POPULATION INFORMATICS

41

Conditional Loops

Code some

POPULATION INFORMATICS

42

Multi Dimensional Arrays

- We only looked at one dimensional arrays
  - SAS: Two dimensional arrays (two indices)
  - array m{4,3} $3. month1-month12;
  - first month of each quarter: m{qtr,1}
  - 4 rows & 3 columns
  - SAS places variables into a two-dimensional array by filling all rows in order, beginning at the upper-left corner of the array (known as row-major order).

where 1<=qtr<=4

month1 (Jan)	month2 (Feb)	month2 (Mar)
month4 (Apr)	month5 (May)	month6 (Jun)
month7 (Jul)	month8 (Aug)	month9 (Sep)
month10 (Oct)	month11 (Nov)	month12 (Dec)

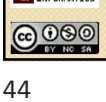
POPULATION INFORMATICS

43

Summary

- Use arrays to recode groups of variables
- Use arrays to create and initialize new groups of variables
- Use arrays to count across a group of variables
- When using arrays/loops you need to look at the code from the perspective of the computer to understand what is happening internally
- Be patient!
  - You will run into many errors when you start writing loops/arrays
  - But practice makes perfect. Practice writing small codes

POPULATION INFORMATICS



44

Use arrays to recode groups of variables

- You have five variables, which were all coded as 99 for refuse to answer
- You want to recode all five variables so that 99 is a missing for analysis

Without using Arrays	Using Arrays

`if var1=99 then var1=.;`
`if var2=99 then var2=.;`
`if var3=99 then var5=.;`
`if var4=99 then var4=.;`
`if var5=99 then var5=.;`
	`array v{*} var1-var5;`
	`do i=1 to dim(v);`
	`  if v{i}=99 then v{i}=.;`
	`end;`

POPULATION INFORMATICS

45

Use arrays to create/initialize groups of variables

- You are creating five new variables to store rates for each month from Jan-May
- You need to initialize all of them to be 0

Without using Arrays	Using Arrays

`jan=1;`
`feb=1;`
`mar=1;`
`apr=1;`
`may=1;`
	`array m{*} jan feb mar apr may;`
	`do i=1 to dim(m);`
	`  m{i}=0;`
	`end;`

POPULATION INFORMATICS

46

Use arrays to count across groups of variables

- You want to know how many assignments were over 90
- Complex if not using arrays
  - Create temporary binary variables for each assignment first
  - Then sum the binary variables

Without using Arrays	Using Arrays

`if assign1>90 then`
`  bassign1=1;`
`if assign2>90 then`
`  bassign2=1;`
`… for all 6 vars …`
`cnt=sum (of assign1-assign6);`
`drop bassign1-bassign6;`
	`*assign1-assign6;`
	`array assign{6};`
	`cnt=0;`
	`do i=1 to dim(assign);`
	`  if assign{i}>90 then`
	`  cnt=cnt+1;`
	`end;`

Population Informatics

47

Algorithms

- Common Idioms
  - Divide & Conquer
  - Iterate
  - Copying
  - Counting
  - Summing
  - Searching
  - Sorting

Population Informatics

48

Reminder

- Review
  - Loops
    - do loops (counting loops)
    - while loops
  - Efficiency concepts
- Assign 3
  - Lab 3 this week
  - Assignment 3 next week
- Read
  - UCLA module (see website)
  - Little SAS book
    - 3.11 Simplifying programs with arrays
    - 3.12 Using Shortcuts to Lists of Variable Names

POPULATION INFORMATICS

BY NC SA

49

POPULATION INFORMATICS

BY NC SA

50