Loops & Arrays

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

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

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

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

Required Reading

- UCLA module
  - http://www.ats.ucla.edu/stat/sas/modules/acrossvars.htm
- 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

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)
do while (expression); 
statements; 
end;

User Efficiency

- **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

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.
Arrays

- 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

SAS: Arrays

- 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;

Lab 3 Objective

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

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

Option 1: 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)
- Review assignment 3 midpoint email together

File name (7 in total)

- kum2.sas (either your code, or commented my code)
- kum2.log
- kum2.htm or kum2.lst
- kum2lab.sas
- kum2lab.log
- kum2lab.lst
- kum2readme.txt

Do Not type text into BB during submission.
Use P1.3 in the comment so I can locate it.
**Counted (Iterative) Loops**

- **Brute Force: Cut & Paste & Tweak**
  - if i = 1 then bever = 1;
  - else if i = 2 then bever = 0;
  - if i = 3 then bever = 1;
  - then bever = 0;
  - then bever = 1;
  - then bever = 0;

- **Using arrays is much more elegant and accurate**:
  - array bever[4] cigever alcever cocever mjever;
  - array bever[4] cigever alcever cocever mjever;
  - do i = 1 to 4;
  - if ever[1] = 1 then bever[i] = 1;
  - else if ever[i] in (0, 2) then bever[i] = 0;
  - end;

- **Even better, more extensible, using arrays**:
  - array bever[4] cigever alcever cocever mjever;
  - array bever[4] cigever alcever cocever mjever;
  - 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;

---

**SAS: for loop statement**

The counted loop solution:

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

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

- **Even better, more extensible, using arrays**:
  - array bever[4] cigever alcever cocever mjever;
  - array bever[4] cigever alcever cocever mjever;
  - 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;
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:
```

---

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:
```

---

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

---

How to figure out new syntax

- Changes over time
- Find a reliable source you like
  - http://support.sas.com/onlinedoc/913/docMainpage.jsp
    - index / do
    - google
      - stata foreach over multiple varlist

---

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.
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

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.

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
    - While: at the top of the loop before the statements in the DO loop have been executed.
    - 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.
  - Stops when
    - While: If the expression is false, the DO loop does not iterate again
    - Until: If the expression is true, the DO loop does not iterate again. The expression is evaluated at the bottom of the loop and if it is true, the loop will not iterate.

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.

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;

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;
A good example for while loop multiple conditions

```haskell
* 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=.; 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;
if (count=4) then year4=2000+idx;
```

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.

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

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) where 1<=qtr<=4
  - 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).

<table>
<thead>
<tr>
<th>month1</th>
<th>month2</th>
<th>month3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>Apr</td>
<td>May</td>
<td>Jun</td>
</tr>
<tr>
<td>Jul</td>
<td>Aug</td>
<td>Sep</td>
</tr>
<tr>
<td>Oct</td>
<td>Nov</td>
<td>Dec</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Without using Arrays</th>
<th>Using Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>if var1=99 then var1=.;</td>
<td>array v{*} var1-var5;</td>
</tr>
<tr>
<td>if var2=99 then var2=.;</td>
<td>do i=1 to dim(v);</td>
</tr>
<tr>
<td>if var3=99 then var5=.;</td>
<td>if v{i}=99 then v{i}=.;</td>
</tr>
<tr>
<td>if var4=99 then var4=.;</td>
<td>end;</td>
</tr>
<tr>
<td>if var5=99 then var5=.;</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Without using Arrays</th>
<th>Using Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>jan=1;</td>
<td>array m{*} jan feb mar apr may;</td>
</tr>
<tr>
<td>feb=1;</td>
<td>do i=1 to dim(m);</td>
</tr>
<tr>
<td>mar=1;</td>
<td>m{i}=0;</td>
</tr>
<tr>
<td>apr=1;</td>
<td>end;</td>
</tr>
<tr>
<td>may=1;</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Without using Arrays</th>
<th>Using Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>if assign1&gt;90 then bassign1=1;</td>
<td>assign{6};</td>
</tr>
<tr>
<td>if assign2&gt;90 then bassign2=1;</td>
<td>cnt=0;</td>
</tr>
<tr>
<td>... for all 6 vars ...</td>
<td>do i=1 to dim(assign);</td>
</tr>
<tr>
<td>cnt=sum (of assign1,assign6);</td>
<td>if assign{i}&gt;90 then</td>
</tr>
<tr>
<td>drop bassign{6};</td>
<td>cnt=cnt+1;</td>
</tr>
<tr>
<td></td>
<td>end;</td>
</tr>
</tbody>
</table>

Algorithms

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

Reminder

- Review
  - Loops
    - do loops (counting loops)
  - while loops
  - Efficiency concepts
  - Assign 3
    - Lab 3 this week
    - Assignment 3 next week
  - Read
    - UCLA module
      - http://www.ats.ucla.edu/stat/sas/modules/acrossvars.htm
    - Little SAS book
      - 3.11 Simplifying programs with arrays
      - 3.12 Using Shortcuts to Lists of Variable Names