Loops & Arrays

efficiency for statements
while statements

Hye-Chung Kum
Population Informatics Research Group
http://research.tamhsc.edu/pinformatics/
http://pinformatics.web.unc.edu/

Vocabulary
- Directory = folder
- Observations = rows = obs
- Variables = columns = var(s)
- where (date<’18jan2004’);
- Line comments
  - * comments;
  - Length limit 256. If you are using it for long lines pay attention to log for messages.

Loops & Arrays

efficiency for statements
while statements

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

Recommended 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 recommended 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)

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

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.
- **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.

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 changes in multiple places

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
```plaintext
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;
```
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;

Counted (Iterative) Loops

do <varindex> = <start> to <stop>
end;

<Body: do some work with varindex>

do <idx> = <start> to <stop> by <step>
end;

<Body: do some work with varindex>
Indentation – helps outline code
Which is more readable?

Why?

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, 
\[ n = \text{dim(ever)} = 4 \]

Why?

```
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;
```

* Hidden Code: 
\[ i = i + 1 \]
* changes each iteration

Inserted Here
if \[ i <= \text{dim(ever)} \]
\[ <\text{jump back to top of loop}> \]
else \[ <\text{exit loop}> \] 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;
```

Even better, more extensible, using arrays:
array ever(4) cigever alcever coceiver mjever snfever;
array bever(4) cigever alcever coceiver mjever bever;
bcocever;

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;

Using arrays is much more elegant and accurate:
array ever(4) cigever alcever coceiver mjever snfever;
array bever(4) cigever alcever coceiver mjever bever;

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;

Even better, more extensible, using arrays:
array ever(4) cigever alcever coceiver mjever;
array bever(4) cigever alcever coceiver 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;

* Hidden Code:
\[ i = i + 1 \]
* changes each iteration

Inserted Here
if \[ i <= \text{dim(ever)} \]
\[ <\text{jump back to top of loop}> \]
else \[ <\text{exit loop}> \] end
How to figure out new syntax

- http://support.sas.com/onlinedoc/913/docMainpage.jsp
  - index / do
- google
  - stata foreach over multiple varlist

Counted 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
  - Until all 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.
Infinite Loops

```plaintext
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

```plaintext
* 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;
      end;
   end;
```

Better to use the for loop

```plaintext
* Initialize variables:
   array rate[*] rate2001 - rate2013;
   count = 0;
   idx = 1;
   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;
      idx = idx + 1;
   end;
```

A good example for while loop multiple conditions

```plaintext
* 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;
      end;
   end;
   if (count=4) then year4=2000+idx;
```

`leave` statement

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

```plaintext
* Initialize variables:
   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;
      end;
   end;
   * Jump out of while loop;
   if (count = 4) then leave;
   idx = idx + 1;
end;
```

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.

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

Conditional Loops

```
do while (expression);
  statements;
end;
```
```
do until (expression);
  statements;
end;
```
```
do while (expression);
  statements;
end;
```
```
do until (expression);
  statements;
end;
```

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).

<table>
<thead>
<tr>
<th>month1 (Jan)</th>
<th>month2 (Feb)</th>
<th>month3 (Mar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>month4 (Apr)</td>
<td>month5 (May)</td>
<td>month6 (Jun)</td>
</tr>
<tr>
<td>month7 (Jul)</td>
<td>month8 (Aug)</td>
<td>month9 (Sep)</td>
</tr>
<tr>
<td>month10 (Oct)</td>
<td>month11 (Nov)</td>
<td>month12 (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 var3=.;</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 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>*assign1-assign6;</td>
</tr>
<tr>
<td>if assign2&gt;90 then bassign2=1;</td>
<td>array assign[6];</td>
</tr>
<tr>
<td>[for all 6 vars ...]</td>
<td>cnt=0;</td>
</tr>
<tr>
<td>cnt=sum (of assign1-assign6);</td>
<td>do i=1 to dim(assign);</td>
</tr>
<tr>
<td>drop bassign1-bassign6;</td>
<td>if assign[i]&gt;90 then</td>
</tr>
<tr>
<td>[for all 6 vars ...]</td>
<td>cnt=cnt+1;</td>
</tr>
<tr>
<td>end;</td>
<td></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
- Due next week
  - Midpoint email
- Read
  - UCLA module
    - http://www.ats.ucla.edu/stat/sas/modules/summvar.htm
  - Little SAS book
    - 3.11 Simplifying programs with arrays
    - 3.12 Using Shortcuts to Lists of Variable Names

Lab

Break? (must start by 11)
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.

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

Guideline for assignment grading (Total of 8)

- Assignment (Total 4)
  - 1: Submitted code that does not run.
  - 2: Mostly running but incorrect.
  - 3: Correct and meets requirements (i.e uses programming constructs required for the assignment)
  - 4: Correct & Elegant. Comments.
- Answers to questions on the assignment (Total 1)
- Midpoint check email (Total 1)
- Lab (Total 2): recommend submitting after one week to get feedback for assignment

Submission

- Collaboration (specify what)
  - Programming/debugging/taught general use
- Submitting answer as readme.txt
- Lab (Total 2): recommend submitting after one week to assess skill for assignment
  - You really should be starting your assignment at least one week before it is due, in order for you to have sufficient time to iterate and seek help when needed.
- Differentiated class: Could have two levels (easy/moderate)
  - Either is fine