Loops & Arrays

efficiency
for statements
while statements

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

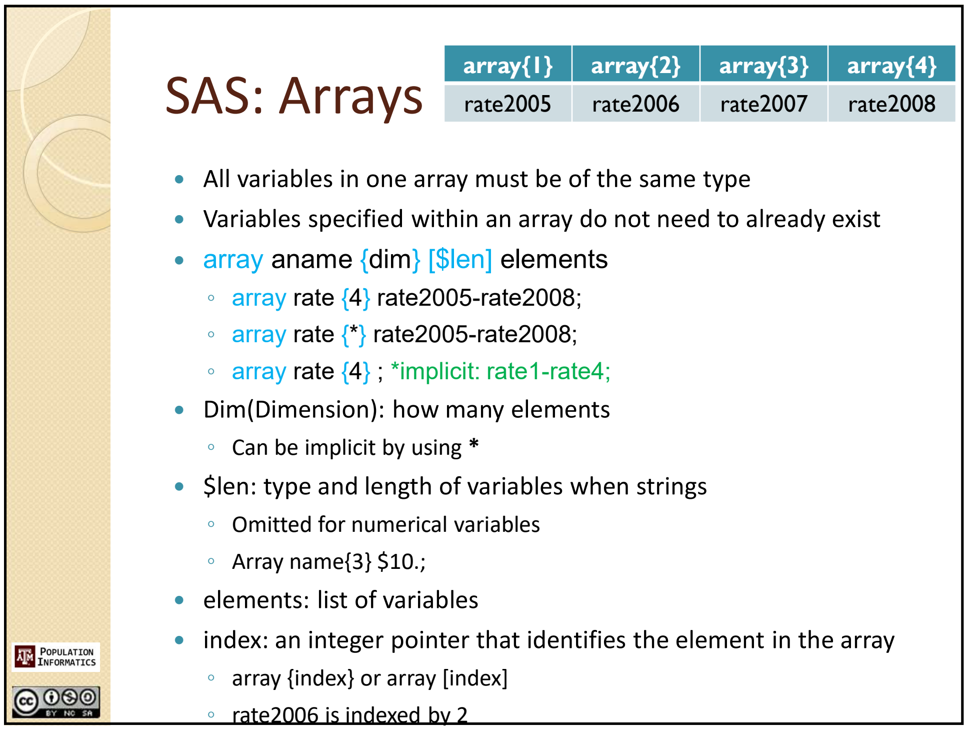
Course URL:
http://pinformatics.org/phpm672

1

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)

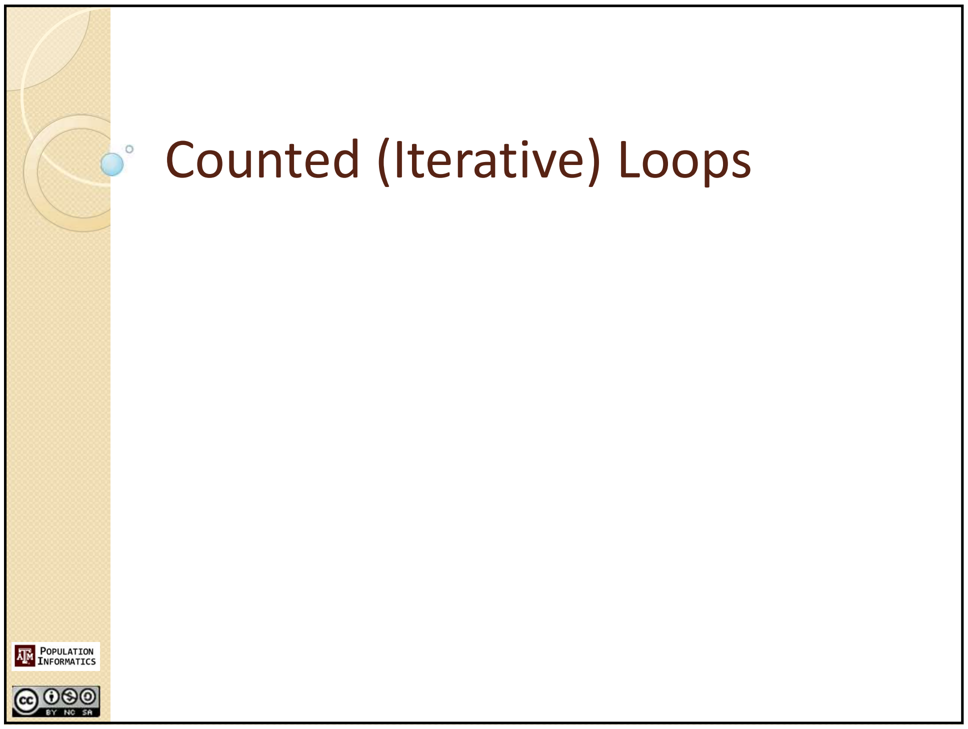
2



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;
- 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



Counted (Iterative) Loops

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;
```

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
```

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;
```

7

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

```
* 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;
```

Indent Why?

8

Conditional Loops

9

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`.

10

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;
```

11

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;
```

12

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 then
      count = count + 1;

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

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

13

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

14

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

15

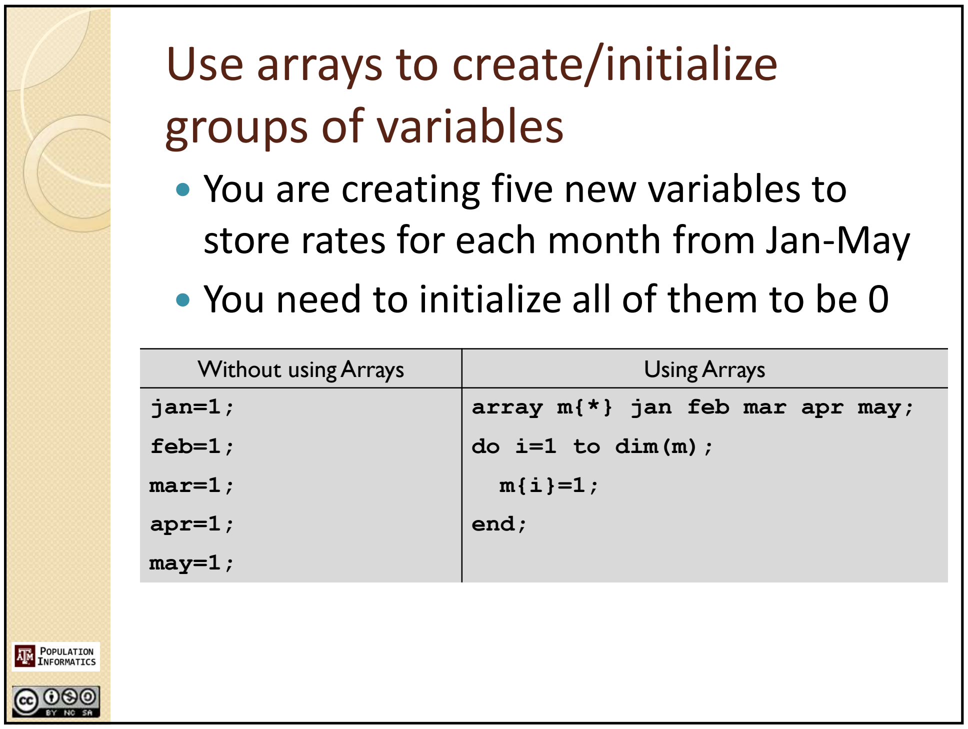
Use arrays to recode groups of variables

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

POPULATION INFORMATICS

16

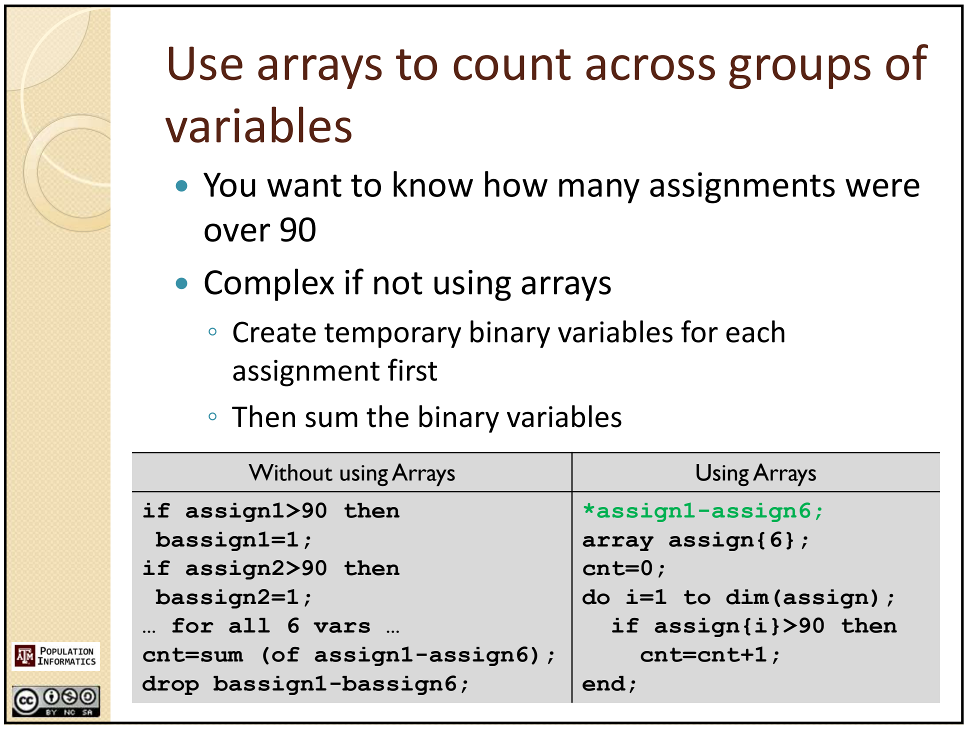


Use arrays to create/initialize groups of variables

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

17



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

18

Algorithms

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