Basic Programming: For Loops and If Statements

DRY vs. WET

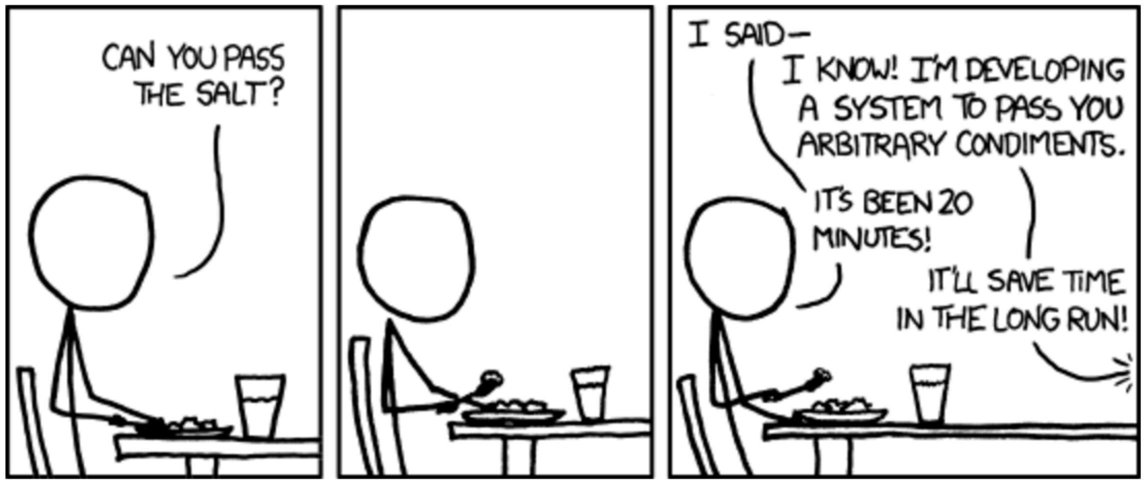
- In programming, there is a DRY credo: **D**on’t **R**epeat **Y**ourself
- Why not?
  - Humans are error-prone
  - Believe it or not, computers are not! The errors they return are caused by humans making mistakes
  - Once a human has invested the abstract problem-solving to solve a task, it makes sense to generalize

WET

- The opposite of DRY is WET:
  - **W**rite **E**verything **T**wice
  - **W**e **E**njoy **T**yping
  - **W**aste **E**veryone’s **T**ime
- This is *probably* too harsh… it’s important not to get paralyzed by finding an optimal solution

WET

- The opposite of DRY is WET:
  - **W**rite **E**verything **T**wice
  - **W**e **E**njoy **T**yping
  - **W**aste **E**veryone's **T**ime
- This is *probably* too harsh… it's important not to get paralyzed by finding an optimal solution


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

For Loops

- For loops are staples of every programming language

- They allow the user to repeat the same action on a list of objects or using a list of values

- There are two commands for looping in Stata:

`foreach` and **`forvalues`**

- We will take a brief look at **`forvalues`** to practice basic looping and then return to foreach for more in-depth examples

Looping Rules

- Stata has a very specific format for for loops:

```
forvalues i = 1/3 { //only comments allowed
   commands ...
}
```

1. The open brace must appear on the same line as the **`forvalues`** or **`foreach`**
2. Nothing besides comments can appear after the open brace
3. The closing brace must appear on a line by itself

forvalues

- What happens when we run this loop?

```
forvalues i = 1/3 { //only comments allowed
    display "Loop Number `i'"
}
```

- The loop will run 3 times, iterating over our values (1, 2, 3) and **`display`**-ing "Loop Number 1", "Loop Number 2", and "Loop Number 3" by placing each value of the

- Notice the Stata iterator is passed into the body of the loop with a very particular method: **` followed by '**
    - **`** is a "grave accent" but is often called a "backtick" or "backquote"
    - **'** is an "apostrophe" or "single quote"

`forvalues` / `foreach`

- `forvalues` is specifically made to iterate over numbers
- It is optimized to be faster than foreach at numbered tasks, but this difference is not substantial for the average user
- `foreach` is a more general looping operator, but has a slightly more complicated syntax
- `foreach` can flexibly iterate over lists of numbers or variables

foreach

```
foreach name of listtype {
    commands . . .
}
```

- The important addition here is the **`listtype`** argument
- This will define the type of iterator we use:
  - `numlist` - list of numbers
  - `varlist` - list of variables
  - `newlist` - list of new variables
- Like **`forvalues`**, we can insert each element of our list using **`` `name' ``** **(backtick + apostrophe)**

foreach - numlist

```
foreach i of numlist 1/7 {
   generate random_variable_`i' = rnormal()
}
```

- In this example, we are creating seven new variables whose values are random pulls from a normal distribution
- This same loop could be made using a `forvalues` call:

```
forvalues j = 1/7 {
   generate random_variable_`i' = rnormal()
}
```

- However, foreach has the flexibility to also iterate over variable lists

foreach - varlist

```
foreach k of varlist make trunk-length {
   ameans `k'
}
```

- In this case, we are iterating over a list of variables and calling their means: `make` and all the variables from `trunk` and `length`
- Using varlist notation like this in scripts is powerful, but depends on the order of your variables
- You can also use the `*` symbol to iterate over all variables

`foreach - newlist`

```
foreach newvar of newlist r_num_1 - r_num_5 {
  generate `newvar' = rpoisson(5)
}
```

- In this case, we are iterating over a list of new variables (`r_num_1`, `r_num_2`, `r_num_3`, `r_num_4`, `r_num_5`) and creating observations based on a random pull from a poisson distribution
- Stata conveniently creates these numbered variables for the user with this **- (hyphen)** notation

`foreach in`

- Finally foreach can be used with `in` to create a shorter list made up of anything the user wishes
- This is useful for iterating over a short number of items, but the added features (like using **-** or **/**) from specifying the type of list (`varlist` or `numlist`) are lost

```
foreach file in autoexpense.dta autosize.dta{
      use `file', clear
      notes: Checked by Cale on 07/26/17
      save `file', replace
 }
```

`If/else` statements

- An `if` statement in a script is different than the `if` qualifier we have used so far
- The purpose of `if/else` statements is to execute code when certain conditions are satisfied (sometimes referred to as “control flow”)
- Often these statements are used inside of for loops to allow a single loop to behave differently based on inputs

`If/else` rules

- Stata has a very specific format for for if/else statements that will be very familiar:

```
if expression { //only comments allowed

    commands

}
```

1. The open brace must appear on the same line as the **`if`** or **`else`**
2. Nothing besides comments can appear after the open brace
3. The closing brace must appear on a line by itself

If/else

- The commands in the body of the `if` statement will only execute if the `expression` evaluates to true (1)

- When the `expression` is anything besides true (1) the body of the `else` statement will execute

```
foreach i of numlist 1/7 {
   if `i' == 4 {
         display “`i' is the best number”
   }
   else {
         display “`i' is a terrible number”
  }
}
```

If/else

- For loops and if/else clauses can also be used to iterate over variables and perform different commands depending on the variable

```
foreach var of varlist headroom trunk weight {
  if "`var'" == "trunk" {
      display "`var' summarize results below"
      summarize `var'
  }
  else {
      display "`var' codebook results below"
      codebook `var'
  }
}
```

Important use of " "

- Note this element of the `if` clause in the previous slide:

```
if "`var'" == "trunk" {
      display “`var' summarize results below”
      summarize `var'
}
```

- Without the quotation marks around **"`var'"** and **"trunk"**, Stata would have checked to see if the *first observation* of our iterating variable and `trunk` were the same

- Therefore, when you want to write an `if / else` statement that checks if the iterator is a particular variable in the dataset, you can think of it as checking whether the ***names*** of the variables are equal

  - To do this, use quotation marks around both:

```
if "`var'" == "trunk"
```

Nested `for` loops

- For loops can also be nested for more advanced behavior:

```
foreach num of numlist 1/3 {
  use auto, clear
  sample `num'
  foreach var of varlist make mpg {
      list `var'
    }
}
```

- What do these for loops do?

Exercises (1)

1. Auto Data

   A. Create a **for loop** in your auto do-file which separately summarizes every variable except for make.

   B. Create a **for loop** in your auto do-file which creates three scatter plots. Price should be on the y-axis in all three, but the x-axis should differ each time: mpg, weight, turn.

   C. Create a **for loop** in your auto do-file which subsets the data based on the values of our price categorical variable and saves these subsets as three separate files.

Exercises (2)

1. For Loop Practice (create a new forloop.do)

   A. Create a for loop that opens our three practice files (auto, titanic, and movies) and shows their notes.

   B. Create a for loop that displays the results of the 8 times tables (8,16,24 etc. up to 8 * 25).

   C. Create five copies of our titanic data file, naming them titanic1, titanic2, etc. However, skip titanic3!