Development Data Boot Camp
Intro to Stata: Basic Programming

Ge Sun

University of Notre Dame

May 18, 2023

Outline

Outline

Intro to “if”

- In some cases, we want to take actions to a sub-sample of the database.
  * Example from yesterday’s exercise:
    What the average population level of villages in the rural area?

Intro to "if"

- In some cases, we want to take actions to a sub-sample of the database.
  - Example from yesterday's exercise:
    What the average population level of villages in the rural area?

    sum Ctot_p if tru == "Rural"
- *if* qualifier restricts the scope of a command to those observations for which the value of the expression is *true*.

Intro to "if"

- In some cases, we want to take actions to a sub-sample of the database.
  - Example from yesterday's exercise:
    What the average population level of villages in the rural area?

    sum Ctot_p if tru == "Rural"
- *if* qualifier restricts the scope of a command to those observations for which the value of the expression is *true*.
- How would a computer express what is true?

Conditional expression

- A conditional expression is a judgement to a statement
  - The statement is true: return 1
  - The statement is false: return 0
- this 1 and 0 are not simple numbers (they are often called "boolean"), they are logic statement, expressing whether the statement is true

Conditional expression

How should we ask Stata to judge a statement?

- greater than : $>$
- smaller than : $<$
- equal to : $==$
- not less than: $>=$
- not greater than: $<=$
- not equal to: $!=$

Digression: "==" vs. "="

- "==" is a logical operation, testing equality
- "=", like many other operating languages, assigns number to another variable

Digression: "==" vs. "="

- "==" is a logical operation, testing equality
- "=", like many other operating languages, assigns number to another variable

Examples:

- $5 == 2 + 3$
- $a = 2 + 3$

Digression: "==" vs. "="

- "==" is a logical operation, testing equality
- "=", like many other operating languages, assigns number to another variable

Examples:

- 5 == 2 + 3
- a = 2 + 3
- gen ill_rate = Cp_ill / Ctot_p
- ill_rate == Cp_ill / Ctot_p

Compounded conditional expressions

What if we need more conditions to pick our sub-sample?

- For more than one restriction, we use and, "& "
  - For example,if we want to explore the income of samples aged between 30 and 40, we should type

```
sum income if age >=30 & age <=40
```

- For more possibilities, we use or, "|"
  - For example, if we want to explore the income of samples who are either younger than 30 or older than 40, we should type

```
sum income if age <=30 | age >=40
```

Compounded conditional expressions

- Generally, $A\&B$ is true if both A and B are true, $A|B$ is true if at least one of A and B is true.
  * $4 < 5$ & $4 ==5$
  * $4 < 5 \mid 4 == 5$

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- *count if age* $< 30 \mid$ *age* > 45

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- *count if age* $< 30 \mid$ *age* > 45
  - 8

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- *list UqNr if hours >=65 & black ==1*

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- *list UqNr if hours >=65 & black ==1*
  - 1021027001901

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- What would you do if you want to know the average earnings per week for female employees that work at least 40 hours a week?

Exercise

	UqNr	Indus_Sep2003	hours	earnings_w~k	age	black	female
1	1011001005301	Manufacturing	45	1250	29	0	0
2	1011001018501	Wholesale and retail trade	64	700	30	0	1
3	1011002015101	Community, social and personal services	48	975	26	1	0
4	1011003007401	Manufacturing	45	2000	44	0	1
5	1011005011001	Community, social and personal services	40	2000	37	0	0
6	1011005011001	Wholesale and retail trade	45	375	29	0	1
7	1011005011001	Transport, storage and communication	70	875	28	0	0
8	1011007004302	Private households	40	212.5	47	1	1
9	1011170010901	Transport, storage and communication	40	750	42	0	1
10	1011171010801	Community, social and personal services	56	2250	25	0	1
11	1011172010801	Community, social and personal services	40	1400	37	0	1
12	1011172017601	Community, social and personal services	40	2000	49	0	1
13	1021015022301	Wholesale and retail trade	45	875	30	0	1
14	1021026002601	Electricity, gas and water supply	54	625	34	1	0
15	1021026006201	Transport, storage and communication	40	375	31	1	1
16	1021026008001	Construction	40	380	34	1	0
17	1021026009801	Construction	45	300	27	1	0
18	1021026011701	Community, social and personal services	45	625	36	1	0
19	1021027001901	Wholesale and retail trade	66	260	36	1	0
20	1021027003601	Construction	45	250	30	1	0

- What would you do if you want to know the average earnings per week for female employees that work at least 40 hours a week?
  - *sum earnings_week if female==1 & hours >= 40*

Outline

Intro to Scalars and Matrices

- Scalars and Matrices are used for storing **numbers** in Stata.
- The information stored in scalars and matrices are different from the "data" we import. And we will not see them in the variable window.
- Scalars and Matrices are useful in storing results of estimation commands: r-class and e-class command

Scalars

- Scalars can store a single number or a single string. Command *scalar* can be used to generate a scalar, for example,

```
scalar a = 2*3
scalar b = "2 times 3 = "
```

- Command *display* is used to display strings and values of scalar expressions:

```
display b a
```

- The most common use of scalars is to store results of estimation commands. (Or use it as a calculator)

Matrices

- Matrices can store several numbers or strings as an array. We use *matrix define* to create a matrix and use *matrix list* to print the matrix:

$$\text{matrix define } A = (1, 2, 3 \backslash 4, 5, 6)$$
$$\text{matrix list } A$$

- We can also display of a specific element of the matrix

$$\text{scalar } c = A[2, 3]$$
$$\text{display } c$$

Intro to Scalars and Matrices

- Scalars and Matrices are used for storing **numbers** in Stata.
- The information stored in scalars and matrices are different from the "data" we import. And we will not see them in the variable window.
- Scalars and Matrices are useful in storing results of estimation commands: r-class and e-class command

Intro to Scalars and Matrices

- Scalars and Matrices are used for storing **numbers** in Stata.
- The information stored in scalars and matrices are different from the "data" we import. And we will not see them in the variable window.
- Scalars and Matrices are useful in storing results of estimation commands: r-class and e-class command

r-class command

- r-class commands: commands that analyze the data but do not estimate parameters, like command *sum*.
- All r-class commands save their results in *r*(), which can be extracted by command *return list*:

```
. sum earnings_week if female==1 & hours >= 40

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
earnings_w~k |         10     1093.75    760.4195       212.5       2250

. return list

scalars:
                  r(N) =  10
              r(sum_w) =  10
               r(mean) =  1093.75
                r(Var) =  578237.8472222222
                 r(sd) =  760.4195205425899
               r(min) =  212.5
               r(max) =  2250
               r(sum) =  10937.5
```

Figure 1: r-class command summarize

r-class command

Type *help sum* to obtain:

```
Stored results

    summarize stores the following in r():

    Scalars
      r(N)            number of observations
      r(mean)         mean
      r(skewness)     skewness (detail only)
      r(min)          minimum
      r(max)          maximum
      r(sum_w)        sum of the weights
      r(p1)           1st percentile (detail only)
      r(p5)           5th percentile (detail only)
      r(p10)          10th percentile (detail only)
      r(p25)          25th percentile (detail only)
      r(p50)          50th percentile (detail only)
      r(p75)          75th percentile (detail only)
      r(p90)          90th percentile (detail only)
      r(p95)          95th percentile (detail only)
      r(p99)          99th percentile (detail only)
      r(Var)          variance
      r(kurtosis)     kurtosis (detail only)
      r(sum)          sum of variable
      r(sd)           standard deviation
```

Figure 2: Interpret r-class command summarize

r-class command

- to exact the results in the r-class command, we use *display* command:

```
display r(mean)
```

- if we want to save one of the results for future calculation, we will use scalar to store the value:

```
scalar mean_earnings = r(mean)
scalar range_earnings = r(max) - r(min)
```

- IMPORTANT: *return list* has to follow the *sum* command to return the statistics of The variable.

e-class command

- e-class commands: commands that store estimate parameters after commands like *reg*.

reg ln_wage_hr age age_sq female black edu

```
. reg ln_wage_hr age age_sq female black edu

      Source |       SS           df       MS      Number of obs   =     6,550
-------------+----------------------------------   F(5, 6544)      =    671.16
       Model |  1867.85901         5  373.571802   Prob > F        =    0.0000
    Residual |  3642.40495     6,544  .556602224   R-squared       =    0.3390
-------------+----------------------------------   Adj R-squared   =    0.3385
       Total |  5510.26396     6,549  .84139013    Root MSE        =    .74606

------------------------------------------------------------------------------
   ln_wage_hr |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
--------------+---------------------------------------------------------------
          age |   .0789691   .0142764     5.53   0.000     .0509828    .1069554
       age_sq |  -.0006551   .0001911    -3.43   0.001    -.0010297   -.0002805
       female |   -.509878   .0190329   -26.79   0.000    -.5471887   -.4725673
        black |  -1.218561   .0414831   -29.37   0.000    -1.299881    -1.13724
          edu |   .0889123   .0025659    34.65   0.000     .0838823    .0939423
        _cons |   .3902671   .2653466     1.47   0.141     -.129899    .9104332
------------------------------------------------------------------------------
```

e-class command

- All e-class commands save their results in *e*(), which can be extracted by command *ereturn list*:

```
. ereturn list

scalars:
                  e(N) =  6550
               e(df_m) =  5
               e(df_r) =  6544
                  e(F) =  671.1647663903854
                 e(r2) =  .3389781362695488
               e(rmse) =  .7460577886363673
                e(mss) =  1867.859008166262
                e(rss) =  3642.404953757752
               e(r2_a) =  .3384730767770897
                 e(ll) =  -7372.208995478296
               e(ll_0) =  -8727.955384072211
               e(rank) =  6

macros:
           e(cmdline) : "regress ln_wage_hr age age_sq female black edu"
             e(title) : "Linear regression"
         e(marginsok) : "XB default"
               e(vce) : "ols"
           e(depvar) : "ln_wage_hr"
               e(cmd) : "regress"
        e(properties) : "b V"
           e(predict) : "regres_p"
             e(model) : "ols"
         e(estat_cmd) : "regress_estat"

matrices:
                  e(b) :  1 x 6
                  e(V) :  6 x 6

functions:
            e(sample)
```

e-class command

```
matrices:
                  e(b) :  1 x 6
                  e(V) :  6 x 6
```

- To extract the coefficient matrix and variance-covariance matrix, we would use:

$$\text{matrix define B} = \text{e(b)}$$
$$\text{matrix define V} = \text{e(V)}$$
$$\text{display V[3,3]}$$
$$\text{scalar var\_3} = \text{V[3,3]}$$

Outline

Macros

- A macro is a string of characters that stands for another string of characters.
  - for example, in the previous exercise, we can use the macro *regressors* in place of a list of regressors, *age age_sq female black edu*.
- Macros can lead to code that is shorter, easier to read, and that can be easily adapted to similar problems.

Macros

- A macro is a string of characters that stands for another string of characters.
  - for example, in the previous exercise, we can use the macro *regressors* in place of a list of regressors, *age age_sq female black edu*.
- Macros can lead to code that is shorter, easier to read, and that can be easily adapted to similar problems.
- Macros can be global or local (how broad this new definition will be applied to)
  - Global macro: across Stata do-files or throughout a Stata session.
  - Local macro: only within a given do-file or in the interactive session.

Global macro

- Global macros are the simplest macro and are very useful.
- Global macros are defined with the *global* command:

```
global regressors age age_sq female black edu
```

- To access what was stored in a global macro, put the character $ in front of the macro name:

```
reg ln_wage_hr $regressors
```

which is equivalent to

```
reg ln_wage_hr age age_sq female black edu
```

How to use global macros

- to create regressor lists for many regression equations.
  * as we have seen in the example, we can use global macros when fitting several models with the same regressor list because they ensure that the regressor list is the same in all instances.
  * In addition, they make it easy to change the regressor list.
- to set constant key parameters through all models.
  * for example, we can set macro *nbreps* the number of bootstrap replications in all models.
- to represent working directory
  * *global macro main_path "/Users/gesun/Desktop/Bootcamp/"* So that we do not need to input the long path name every time! (That is also why it is important to keep a clear and reasonable sub-folder structure)

How to use global macros

- to create regressor lists for many regression equations.
  * as we have seen in the example, we can use global macros when fitting several models with the same regressor list because they ensure that the regressor list is the same in all instances.
  * In addition, they make it easy to change the regressor list.
- to set constant key parameters through all models.
  * for example, we can set macro *nbreps* the number of bootstrap replications in all models.
- to represent working directory
  * *global macro main_path "/Users/gesun/Desktop/Bootcamp/"* So that we do not need to input the long path name every time! (That is also why it is important to keep a clear and reasonable sub-folder structure)
- You need to be very careful when using global macro

Local macros

- Local macro can be accessed only within a given do-file or in the interactive session, which can be defined using *local* command:

```
local regressors age age_sq female black edu
```

- To access the content of local macros, enclose the macro name in a single quotes:

```
local regressors age age_sq female black edu
reg ln_wage_hr `regressors'
```

- Note that the left quote is not a single quotation mark, it is a Backtick located at the upper left.

Digression: Should I use = to define a macro?

- We can define a macro in two ways:

  local i 1

  local i=1

  and in this case, these two definition are equivalent.

- However, when the right hand side is an expression, the macro will the be defined as the evaluation of the expression with "=", and it will be defined as the expression itself when there is no "=".

  local i 1 + 1 // i will be replaced by 1+1

  local i = 1 +1 // i will be replaced by 2

- Type the following command to see the difference:

  display "'i'"

Global or local?

- Local macros apply only to the current program and have the advantage of no potential conflict with other programs.
- If you have a complex document structure, like many Do-files, local macros are preferred to global macros. Always be careful when using global macros, because the use of macro may lead to implicit bugs.
- Local macros are especially useful for programming in Stata:
  - Writing your own function.
  - Loops

Additional tips about applying macros

Scalar or Macro ?

- Using a scalar will usually be faster than using a macro, because a macro requires convention into and out of internal binary representation.
- However, scalar is dropped whenever “clear all” (not “clear”) is used.

Additional tips about applying macros

Scalar or Macro ?

- Using a scalar will usually be faster than using a macro, because a macro requires convention into and out of internal binary representation.
- However, scalar is dropped whenever “clear all” (not “clear”) is used.
- Macro will continue to exist even after “clear all” (be careful!)

* Another small thing is if you write codes in the dofile instead of the interactive command line, you need to run the local definition command along with other commands that access the local definition.(That is why local does not have that much trouble.)

Outline

Introduction to Loops

- Loops provide a way to repeat the same command many times.
  * Example:
    Suppose we have South Bend monthly average temperature data: *tem1, tem2, tem3 tem4, ..., tem12*, that documents the temperature in Celsius. We want to create 12 new variables that document the temperature in Fahrenheit.

Introduction to Loops

- Loops provide a way to repeat the same command many times.
  * Example:
    Suppose we have South Bend monthly average temperature data: *tem1, tem2, tem3 tem4, ..., tem12*, that documents the temperature in Celsius. We want to create 12 new variables that document the temperature in Fahrenheit.
  * Possible solutions:

$$\begin{aligned} \text{gen tem1\_F} &= (\text{tem1 * 1.8}) + 32 \\ \text{gen tem2\_F} &= (\text{tem2 * 1.8}) + 32 \\ &\dots \\ \text{gen tem12\_F} &= (\text{tem12 * 1.8}) + 32 \end{aligned}$$

Introduction to Loops

- Loops provide a way to repeat the same command many times.
  * Example:
    Suppose we have South Bend monthly average temperature data: *tem1, tem2, tem3 tem4, ..., tem12*, that documents the temperature in Celsius. We want to create 12 new variables that document the temperature in Fahrenheit.
  * Possible solutions:

$$\begin{aligned} \text{gen tem1\_F} &= (\text{tem1 * 1.8}) + 32 \\ \text{gen tem2\_F} &= (\text{tem2 * 1.8}) + 32 \\ &\dots \\ \text{gen tem12\_F} &= (\text{tem12 * 1.8}) + 32 \end{aligned}$$

- Loops can help us with this task by just one command
  * really makes our life easier
  * reduce the possibility to make mistakes

Digression: Programming

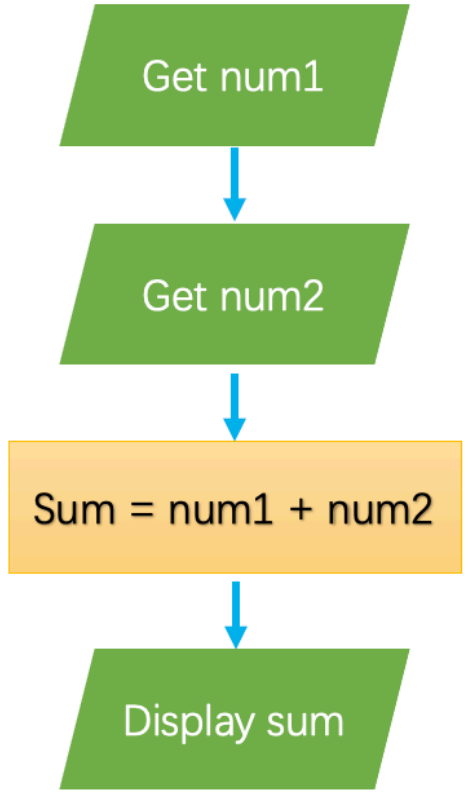
Three ways that a code is executed:

- Sequential
- Selection
- Iteration

Digression: Programming

Three ways that a code is executed:

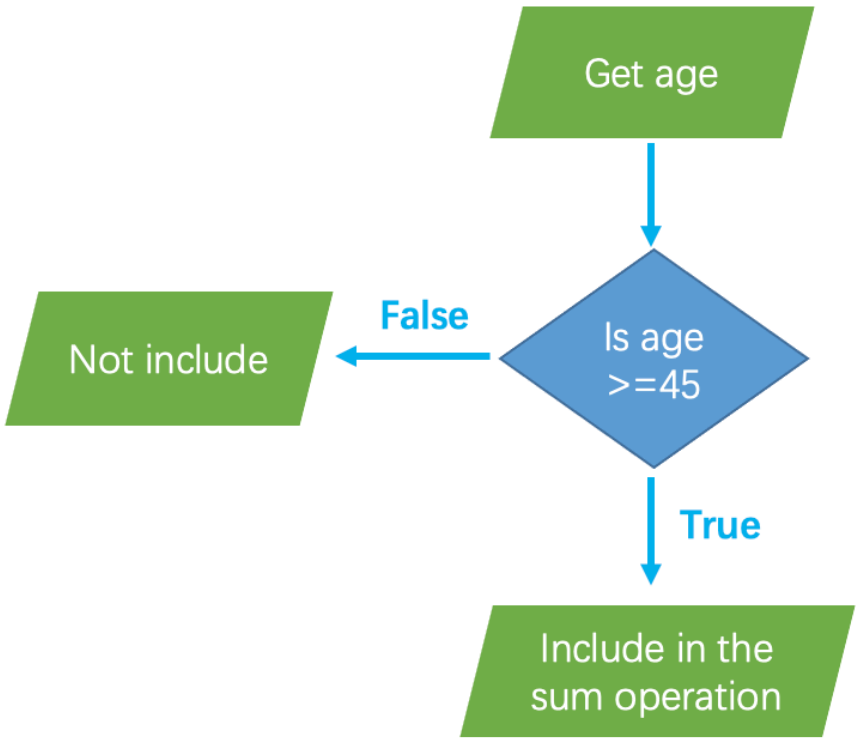
- Sequential



Digression: Programming

Three ways that a code is executed:

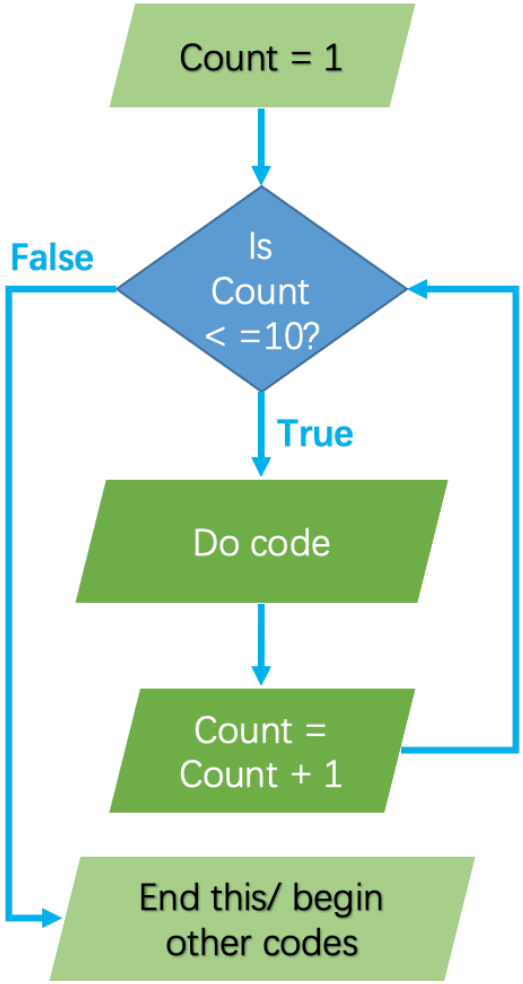
- Selection



Digression: Programming

Three ways that a code is executed:

- Iteration



Digression: Programming

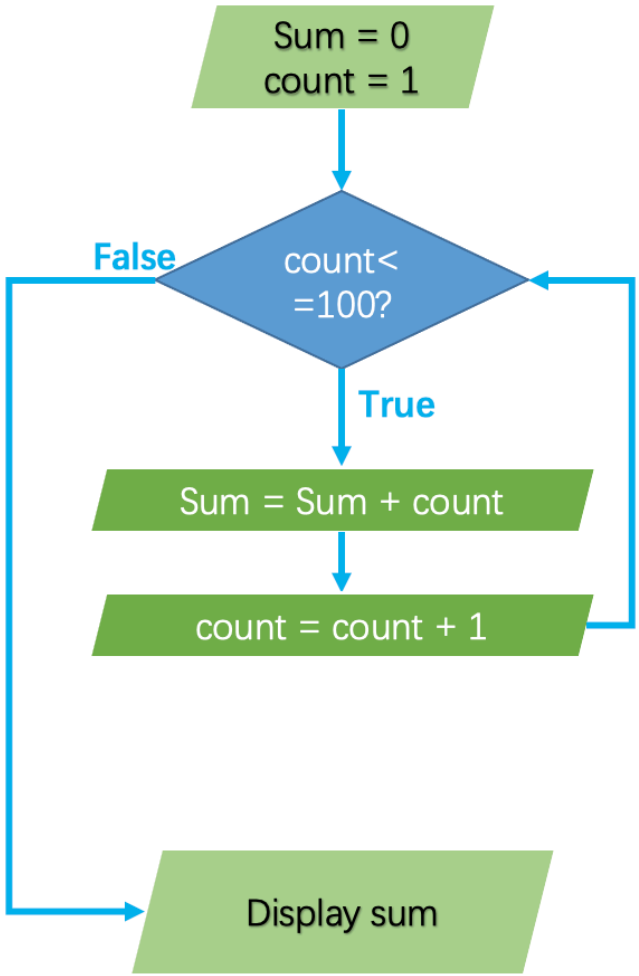
Let's try some looping mind game!

Digression: Programming

- How to ask a computer to sum 1 to 100?

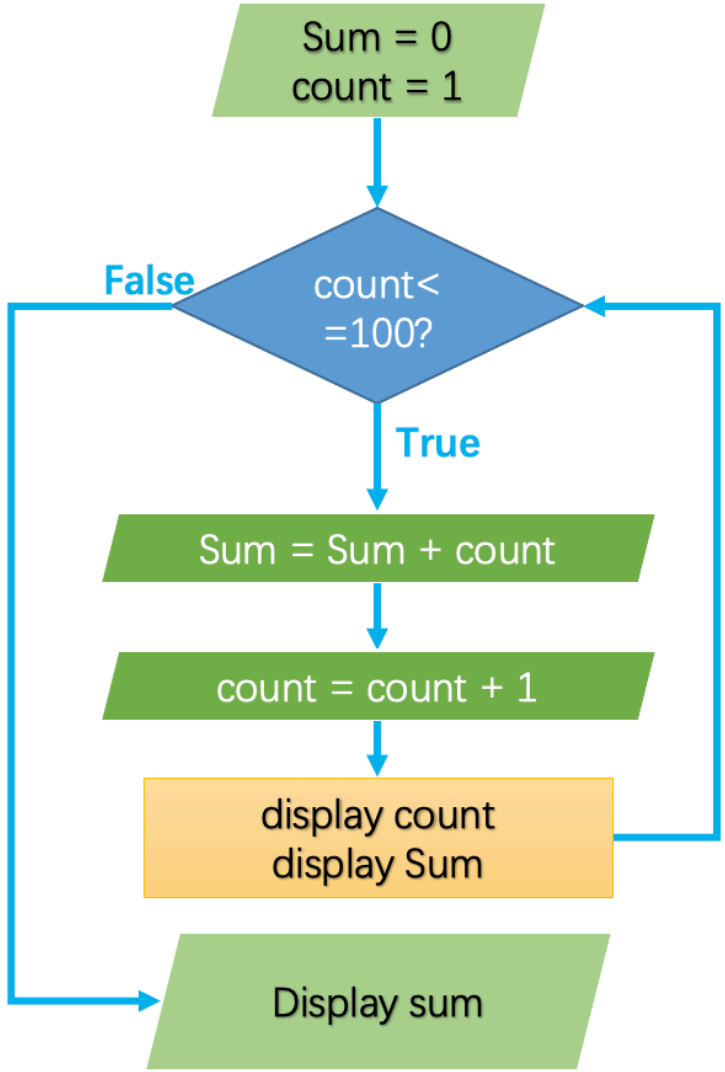
Digression: Programming

- How to ask a computer to sum 1 to 100?



Digression: Programming

- How to ask a computer to sum 1 to 100?



Digression: Programming

- This is a Stretch Exercise!
  How do you use computer to find all the divisors to 187?

Introduction to Loops

- The syntax of loops in Stata is complex and hard to remember exactly. My recommendation is that read the document to find how to write loops properly whenever you need to use loops in your code.

Introduction to Loops

- The syntax of loops in Stata is complex and hard to remember exactly. My recommendation is that read the document to find how to write loops properly whenever you need to use loops in your code.
- There are three loops commands in Stata:
  1. *foreach*: loops over items in a list.
  2. *forvalue*: loops over consecutive values of numbers.
  3. *while*: loops continues until a user-specified condition is not met.
     * The first two are intuitive, the *while* is more like loop we just talked about

Command: foreach

- Command *foreach* can loop over items in a list of variable names or number.
- Suppose we have a list of variables named var_1, var_2, var_3, and var_4, and we want to calculate the sum of these four variables:

```
gen sum_var = 0
foreach var of varlist var_1 var_2 var_3 var_4 {
    replace sum_var = sum_var + `var'
}
```

- In the *foreach* loop, we refer to each variable in the variable list varlist by the local macro named var.

Command: forvalues

- Command *forvalues* loops over consecutive values. Consider the same example in the previous slides:

```
replace sum_var = 0
forvalues i = 1 / 4{
    replace sum_var = sum_var + var_‘i’
}
```

- In the *forvalues* loop, we refer to each variable via their indexes, local macro ‘i’. As ‘i’ goes from 1 to 4, we add the variable var_i to variable sum_var one by one.

Command: while

- Command *forvalues* loops over consecutive values. Consider the same example in the previous slides:

```
local i 1
replace sum_var = 0
while `i'<=4 {
    replace sum_var = sum_var + var_`i'
    local i = `i'+1
}
```

- In the following code, the local macro i is initialized to 1 and then incremented by 1 in each loop.
- Command *while* is more flexible than *foreach* and *forvalues*.

When to use Loops?

- to generate similar variables
- to recode a bunch of variables
- to apply the same bunch of codes to different datasets
- to append data for different years

When you go to copy and paste any commands to repeat them in your do-file, you should be asking yourself: "Should I be using a loop?" The answer is likely YES!