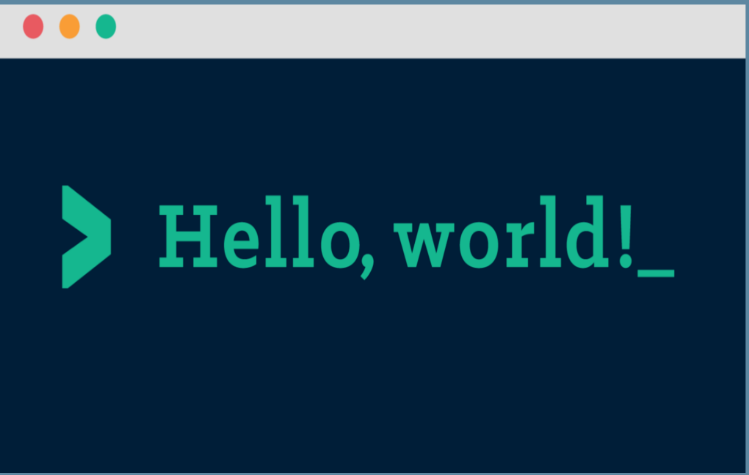
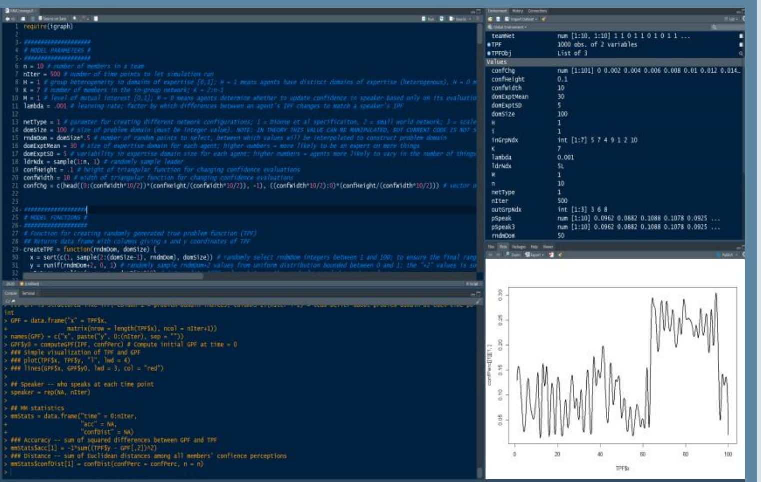




```
require(igraph)

######################
# MODEL PARAMETERS #
######################
n = 10 # number of members in a team
nIter = 500 # number of time points to let simulation run
H = 1 # group heterogeneity in domains of expertise [0,1]; H = 1 means agents have distinct domains of expertise (heterogeneous), H = 0 m
K = 7 # number of members in the in-group network; K = 2:n-1
M = 1 # level of mutual interest [0,1]; M = 0 means agents determine whether to update confidence in speaker based only on its evaluatio
lambda = .001 # learning rate; factor by which differences between an agent's IPF changes to match a speaker's IPF

netType = 1 # paramter for creating different network configurations; 1 = Dionne et al specificaiton, 2 = small world network; 3 = scale
domSize = 100 # size of problem domain (must be integer value). NOTE: IN THEORY THIS VALUE CAN BE MANIPULATED, BUT CURRENT CODE IS NOT S
rndmDom = domSize*.5 # number of random points to select, between which values will be interpolated to construct problem domain
domExptMean = 30 # size of expertise domain for each agent; higher numbers = more likely to be an expert on more things
domExptSD = 5 # variability in expertise domain size for each agent; higher numbers = agents more likely to vary in the number of things
ldrNdx = sample(1:n, 1) # randomly sample leader
confHeight = .1 # height of triangular function for changing confidence evaluations
confWidth = 10 # width of triangular function for changing confidence evaluations
confChg = c(head((0:(confWidth*10/2))*(confHeight/(confWidth*10/2)), -1), ((confWidth*10/2):0)*(confHeight/(confWidth*10/2))) # vector o


####################
# MODEL FUNCTIONS #
####################
# Function for creating randomly generated true problem function (TPF)
## Returns data frame with columns giving x and y coordinates of TPF
createTPF = function(rndmDom, domSize) {
  x = sort(c(1, sample(2:(domSize-1), rndmDom), domSize)) # randomly select rndmDom integers between 1 and 100; to ensure the final rang
  y = runif(rndmDom+2, 0, 1) # randomly sample rndmDom+2 values from uniform distribution bounded between 0 and 1; the "+2" values is so
```

```
int
> GPF = data.frame("x" = TPF$x,
+                  matrix(nrow = length(TPF$x), ncol = nIter+1))
> names(GPF) = c("x", paste("y", 0:(nIter), sep = ""))
> GPF$y0 = computeGPF(IPF, confPerc) # Compute initial GPF at time = 0
> ### Simple visualization of TPF and GPF
> ### plot(TPF$x, TPF$y, "l", lwd = 4)
> ### lines(GPF$x, GPF$y0, lwd = 3, col = "red")
>
> ## Speaker -- who speaks at each time point
> speaker = rep(NA, nIter)
>
> ## MM statistics
> mmStats = data.frame("time" = 0:nIter,
+                      "acc" = NA,
+                      "confDist" = NA)
> ### Accuracy -- sum of squared differences between GPF and TPF
> mmStats$acc[1] = -1*sum((TPF$y - GPF[,2])^2)
> ### Distance -- sum of Euclidean distances among all members' confience perceptions
> mmStats$confDist[1] = confDist(confPerc = confPerc, n = n)
>
```

```
 teamNet       num [1:10, 1:10] 1 1 0 1 1 0 1 0 1 1 ...
●TPF           1000 obs. of 2 variables
●TPFObj        List of 3
Values
 confChg       num [1:101] 0 0.002 0.004 0.006 0.008 0.01 0.012 0.014…
 confHeight    0.1
 confWidth     10
 domExptMean   30
 domExptSD     5
 domSize       100
 H             1
 i             1
 inGrpNdx      int [1:7] 5 7 4 9 1 2 10
 K             7
 lambda        0.001
 ldrNdx        5L
 M             1
 n             10
 netType       1
 nIter         500
 outGrpNdx     int [1:3] 3 6 8
 pSpeak        num [1:10] 0.0962 0.0882 0.1088 0.1078 0.0925 ...
 pSpeak3       num [1:10] 0.0962 0.0882 0.1088 0.1078 0.0925 ...
 rndmDom       50
```

Hello World!

Tools, Techniques, and Getting Started with Computational Modeling

36th Annual Conference of the Society for Industrial & Organizational Psychology

James A. Grand

University of Maryland

WHAT IS A COMPUTATIONAL MODEL?

- FORMAL & ALGORITHMIC DESCRIPTION OF HOW A PHENOMENON UNFOLDS OVER TIME
  - "WHAT HAPPENS" TO PRODUCE "WHAT WE OBSERVE"

Formal

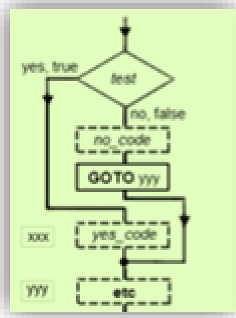
- Declarative logic and mathematical equations used to convey how, when, and why variables change

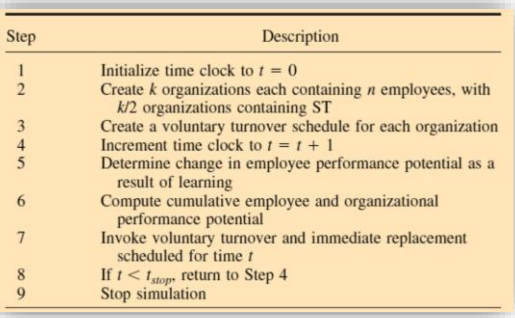
$$\begin{aligned} &IF\ [goal_1 > goal_2]: \\ &\qquad THEN\ [choice_t = goal_1] \\ &\qquad ELSE\ [choice_t = goal_2] \end{aligned}$$

$$goal_n = \frac{1}{1 + e^{-\lambda * (X_t - X_{t-1})}}$$

Algorithmic

- Proposed sequence in which actions/events occur over time or in response to other actions/events

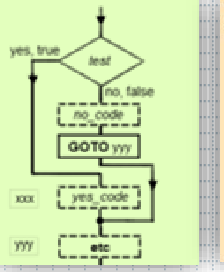


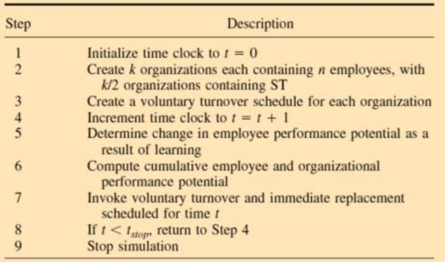


Step	Description
1	Initialize time clock to $t = 0$
2	Create k organizations each containing n employees, with $k/2$ organizations containing ST
3	Create a voluntary turnover schedule for each organization
4	Increment time clock to $t = t + 1$
5	Determine change in employee performance potential as a result of learning
6	Compute cumulative employee and organizational performance potential
7	Invoke voluntary turnover and immediate replacement scheduled for time t
8	If $t < t_{stop}$, return to Step 4
9	Stop simulation

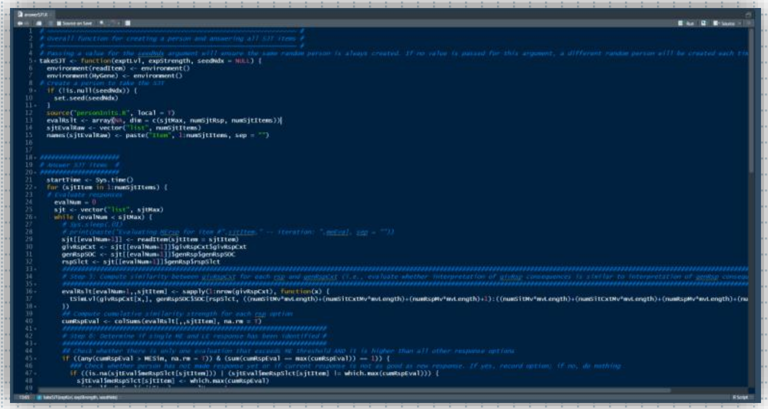
WHAT IS A COMPUTATIONAL MODEL?

- FORMAL & ALGORITHMIC DESCRIPTION OF HOW A PHENOMENON UNFOLDS OVER TIME
  - TYPICALLY TRANSLATE MODEL INTO COMPUTER CODE THAT CAN ENACT MECHANISMS FOR US





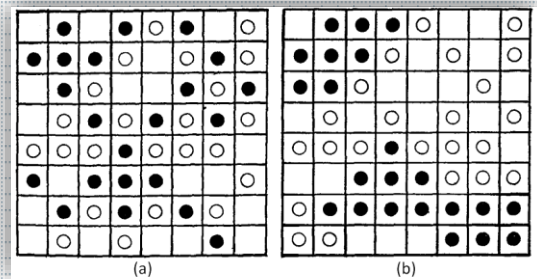
Step	Description
1	Initialize time clock to $t = 0$
2	Create k organizations each containing n employees, with $k/2$ organizations containing ST
3	Create a voluntary turnover schedule for each organization
4	Increment time clock to $t = t + 1$
5	Determine change in employee performance potential as a result of learning
6	Compute cumulative employee and organizational performance potential
7	Invoke voluntary turnover and immediate replacement scheduled for time t
8	If $t < t_{stop}$, return to Step 4
9	Stop simulation



```
# ------------------------------------------------------------------- #
# Overall function for creating a person and answering all SJT items #
# ------------------------------------------------------------------- #
# Passing a value for the seedNdx argument will ensure the same random person is always created. If no value is passed for this argument, a different random person will be created each ti
takeSJT <- function(exptLvl, expStrength, seedNdx = NULL) {
  environment(readItem) <- environment()
  environment(HyGene) <- environment()
# Create a person to take the SJT
  if (!is.null(seedNdx)) {
    set.seed(seedNdx)
  }
  source("personInits.R", local = T)
  evalRslt <- array(NA, dim = c(sjtMax, numSjtRsp, numSjtItems))
  sjtEvalRaw <- vector("list", numSjtItems)
  names(sjtEvalRaw) <- paste("Item", 1:numSjtItems, sep = "")


#######################
# Answer SJT items  #
#######################
  startTime <- Sys.time()
  for (sjtItem in 1:numSjtItems) {
  # Evaluate responses
    evalNum = 0
    sjt <- vector("list", sjtMax)
    while (evalNum < sjtMax) {
      # Sys.sleep(.01)
      # print(paste("Evaluating MErsp for item #",sjtItem," -- iteration: ",numEval, sep = ""))
      sjt[[evalNum+1]] <- readItem(sjtItem = sjtItem)
      givRspCxt <- sjt[[evalNum+1]]$givRspCxt$givRspCxt
      genRspSOC <- sjt[[evalNum+1]]$genRsp$genRspSOC
      rspSlct <- sjt[[evalNum+1]]$genRsp$rspSlct
      ##########################################################################################################
      # Step 3: Compute similarity between givRspCxt for each rsp and genRspCxt (i.e., evaluate whether interpretation of givRsp consequences is similar to interpretation of genRsp consequ
      ##########################################################################################################
      evalRslt[evalNum+1,,sjtItem] <- sapply(1:nrow(givRspCxt), function(x) {
        tSimLvl(givRspCxt[x,], genRspSOC$SOC[rspSlct, ((numSitMv*mvLength)+(numSitCxtMv*mvLength)+(numRspMv*mvLength)+1):((numSitMv*mvLength)+(numSitCxtMv*mvLength)+(numRspMv*mvLength)+(nu
      })
      ## Compute cumulative similarity strength for each rsp option
      cumRspEval <- colSums(evalRslt[,,sjtItem], na.rm = T)
      ##########################################################################
      # Step 6: Determine if single ME and LE response has been identified #
      ##########################################################################
      ## Check whether there is only one evaluation that exceeds ME threshold AND it is higher than all other response options
      if ((any(cumRspEval > MESim, na.rm = T)) & (sum(cumRspEval == max(cumRspEval)) == 1)) {
        ### Check whether person has not made response yet or if current response is not as good as new response. If yes, record option; if no, do nothing
        if ((is.na(sjtEval$meRspSlct[sjtItem])) | (sjtEval$meRspSlct[sjtItem] != which.max(cumRspEval))) {
          sjtEval$meRspSlct[sjtItem] <- which.max(cumRspEval)
```

  - BUT NOTE THIS IS FOR CONVENIENCE/EFFICIENCY, NOT BECAUSE IT IS REQUIRED!


(a)
(b)

Thomas Schelling won a Nobel Prize running computational models on a checkerboard! (sort of…)

Schelling, T.C. (1974). On the ecology of micromotives. In R. Marris (Ed.), *The Corporate society.* (pp. 19-64). London: Macmillan

Getting Started in Two Steps

Step 1 — Train yourself to think in terms of actors, not factors

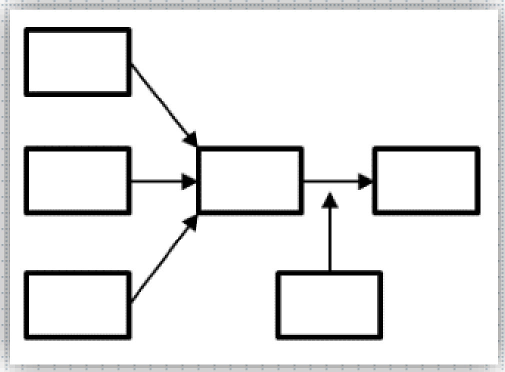
Step 2 — Learn basic programming concepts

- People perceive most important skills for computational modeling are programming and/or quantitative
- But learning how to think computationally is <u>much more important</u>...and very different than how most of us were trained

Step 1

Train yourself to think in terms of actors, not factors

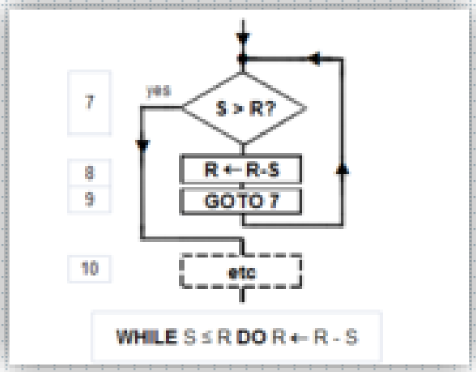
"Factor" Thinking



- Causality = consistent covariation
- Stable relationships among "static" variables
- Statistical

- What are the antecedent and outcome variables?
- How strong is the covariation?
- What variables account for variance in other variables?

"Actor" Thinking



- Causality = Generative mechanisms
- Functional relationships that link "dynamic" variables
- Computational

- What happens to individuals?
- How do individuals think, feel, and react?
- Which and how do actions/events unfold?

Actor vs. Factor Thinking

Macy, M.W., & Willer, R. (2002). From factors to actors: Computational sociology and agent-based modeling. *Annual Review of Sociology, 28*, 143-166.

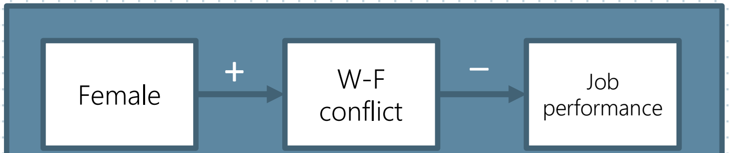
Actor vs. Factor Thinking

Step 1

Train yourself to think in terms of actors, not factors

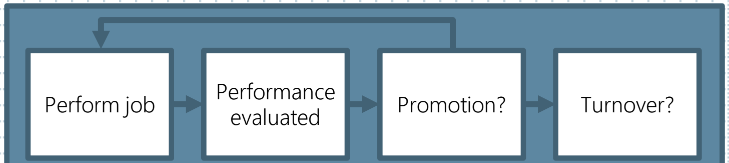
Why are women underrepresented in senior leadership positions?

"Factor" Thinking



- Inference: Females experience greater work-family conflict which reduces performance and likelihood of promotion
- Causal mechanism: Career delays? Lack of opportunities? Discrimination/bias?

"Actor" Thinking



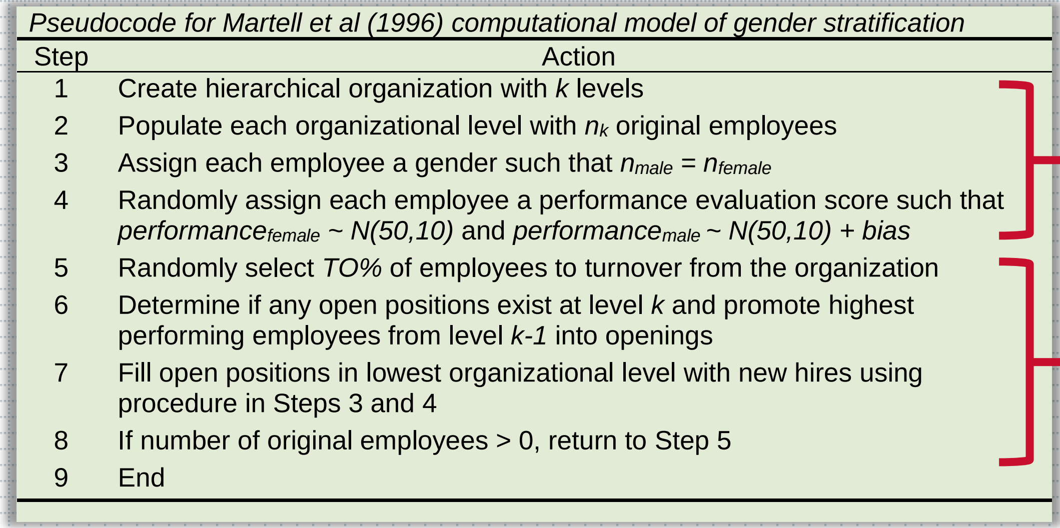
- Inference: Male vs. female performance is evaluated differently leading to different promotion rates
- Causal mechanism: Bias in performance evaluations

Step 1

Train yourself to think in terms of actors, not factors

- PSEUDOCODE → THE "BOX AND ARROWS" OF COMPUTATIONAL THINKING
  - PROPOSED SEQUENCE OF STEPS FOR WHAT OCCURS
  - WHERE DOES THIS COME FROM?
    - » THEORY
    - » OBSERVATION
    - » EMPIRICAL RESULTS…SOMETIMES
    - » "DISCIPLINED IMAGINATION" (WEICK, 1989)

I may spend days or weeks developing the logic, functions, and structure of the pseudocode before I even consider coding!



Pseudocode for Martell et al (1996) computational model of gender stratification

Step	Action
1	Create hierarchical organization with *k* levels
2	Populate each organizational level with n_k original employees
3	Assign each employee a gender such that $n_{male} = n_{female}$
4	Randomly assign each employee a performance evaluation score such that $performance_{female}$ ~ *N(50,10)* and $performance_{male}$ ~ *N(50,10) + bias*
5	Randomly select *TO%* of employees to turnover from the organization
6	Determine if any open positions exist at level *k* and promote highest performing employees from level *k-1* into openings
7	Fill open positions in lowest organizational level with new hires using procedure in Steps 3 and 4
8	If number of original employees > 0, return to Step 5
9	End

INITIALIZATION

WHAT DOES THE "WORLD" LOOK LIKE?

MODEL

WHAT HAPPENS? WHEN? HOW?

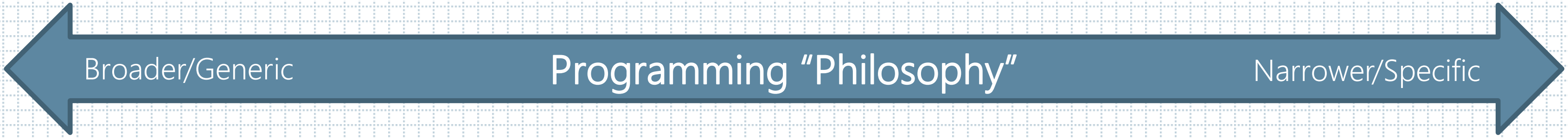
ACTOR VS. FACTOR THINKING

Martell, R.F., Lane, D.M., & Emrich, C. (1996). Male-female differences: A computer simulation. *American Psychologist, 51*, 157-158.

Step 2

Learn basic programming concepts

- Many choices when it comes to modeling software:



Matlab

Python

Java

NetLogo

- Do anything & everything
- Only syntax/coding
- Steeper learning curve (maybe?)

- Do particular things well
- Syntax & ”point & click” hybrid
- Shallower learning curve (maybe?)

Programming Fundamentals 101

Step 2

Learn basic programming concepts

- How do I choose? Which should I learn?

Are you comfortable with syntax-based software for statistics (R, Matlab, Python)?

No & not interested

Yes

Do you know the “type” of model you want to run? (agent-based, system dynamics, neural network, etc.)

Yes & it’s the only type I’m interested in

No, not sure, or don’t care

Broader/Generic

Matlab

Python

Java

Ultimately, this choice is not that important…

Just pick one and start learning!

Narrower/Specific

NetLogo

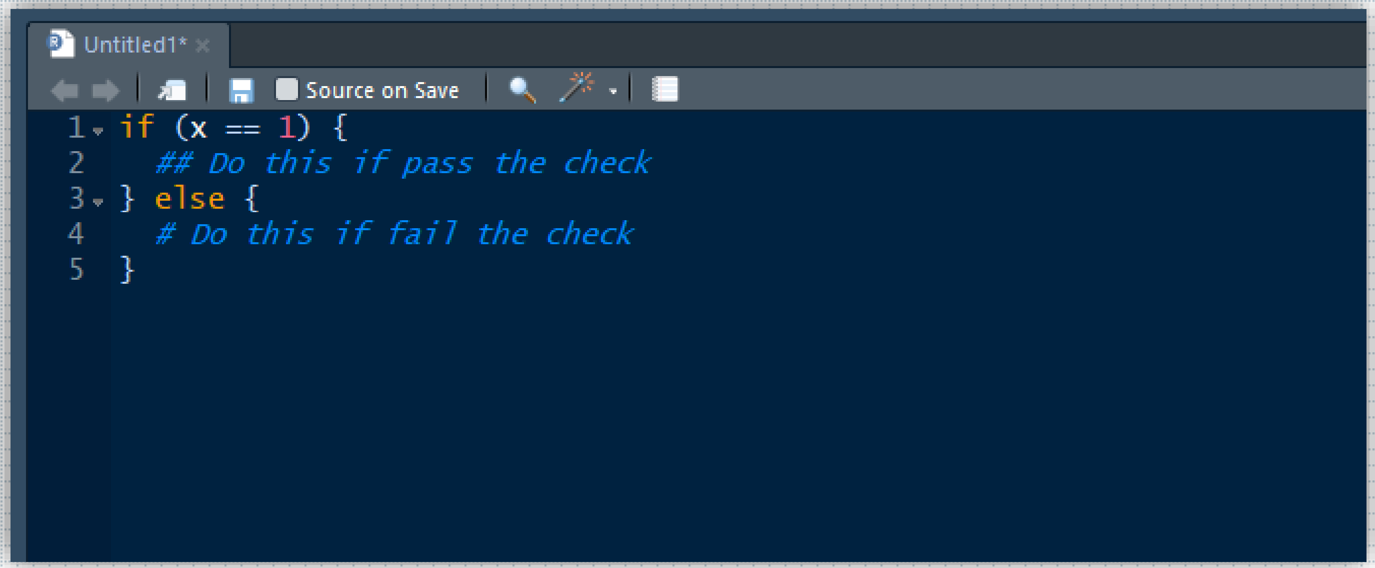
Programming Fundamentals 101

Step 2

Learn basic programming concepts

1 IF-ELSE

- THREE MUST-LEARN PROGRAMMING CONCEPTS
  - IF-ELSE STATEMENTS
    - *BRANCHING* → EXECUTE CODE BASED ON WHETHER SOMETHING IS TRUE OR FALSE
    - E.G., =, ≠, <, >, ≤, ≥



```
if (x == 1) {
  ## Do this if pass the check
} else {
  # Do this if fail the check
}
```

PROGRAMMING FUNDAMENTALS 101

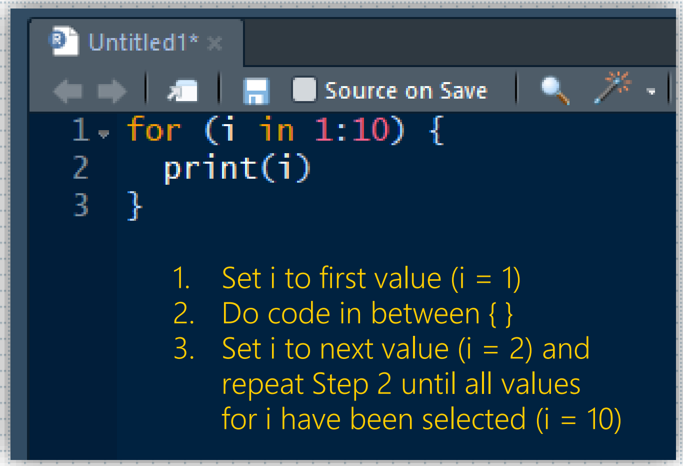
Step 2

Learn basic programming concepts

1 If-Else

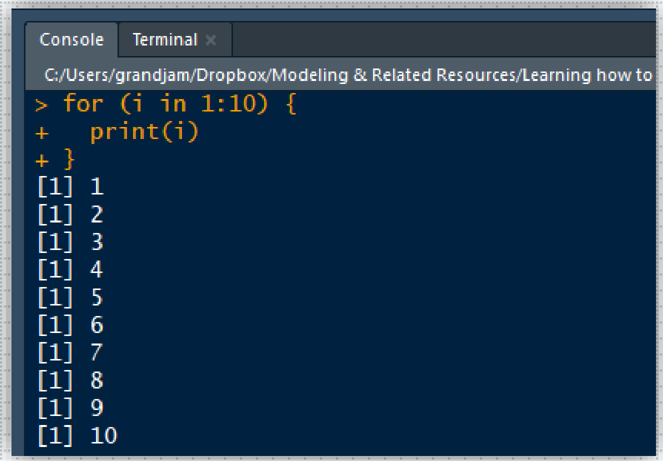
2 Loops

- Three must-learn programming concepts
  - Loops
    - » *Sequence* → execute steps in code multiple times in a row
    - » For loop: Execute code a specific number of times
      1. Use an *iterator* that takes on a set of pre-determined values
      2. Run the code for each value of the iterator
         A. e.g, 1-10, number of rows in a matrix



```
for (i in 1:10) {
  print(i)
}
```

1. Set i to first value (i = 1)
2. Do code in between { }
3. Set i to next value (i = 2) and repeat Step 2 until all values for i have been selected (i = 10)



```
C:/Users/grandjam/Dropbox/Modeling & Related Resources/Learning how to
> for (i in 1:10) {
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Programming Fundamentals 101

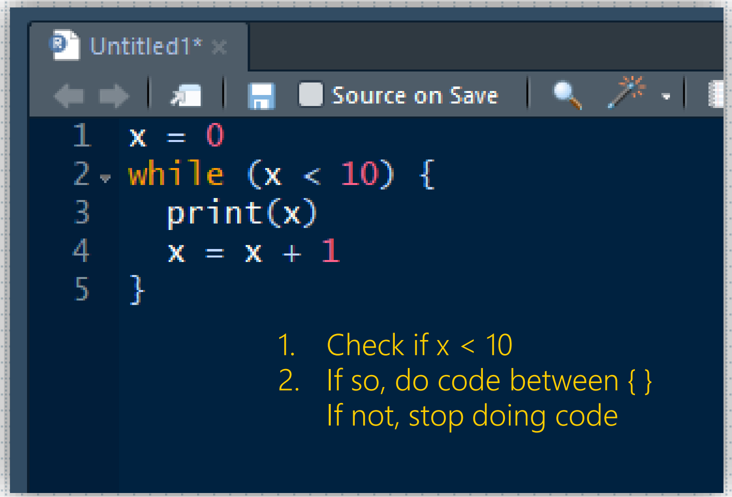
Step 2

Learn basic programming concepts

1 IF-ELSE

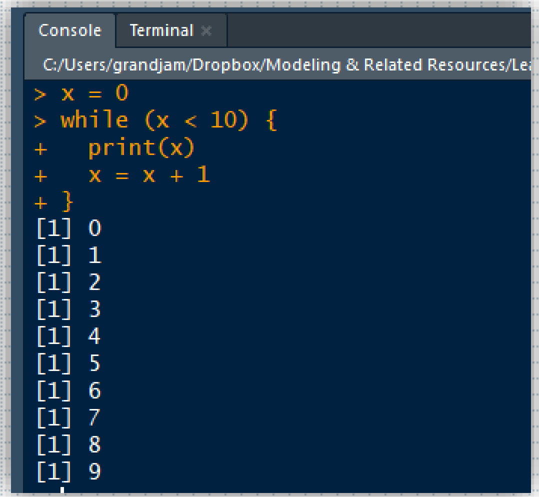
2 LOOPS

- THREE MUST-LEARN PROGRAMMING CONCEPTS
  - LOOPS
    - » *SEQUENCE* → EXECUTE STEPS IN CODE MULTIPLE TIMES IN A ROW
    - » WHILE LOOP: EXECUTE CODE UNTIL A CONDITION IS SATISFIED
      1. KEEP RUNNING CODE WHILE SOMETHING IS TRUE



```
x = 0
while (x < 10) {
  print(x)
  x = x + 1
}
```

1. Check if x < 10
2. If so, do code between { }
   If not, stop doing code



```
> x = 0
> while (x < 10) {
+   print(x)
+   x = x + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

PROGRAMMING FUNDAMENTALS 101

Step 2

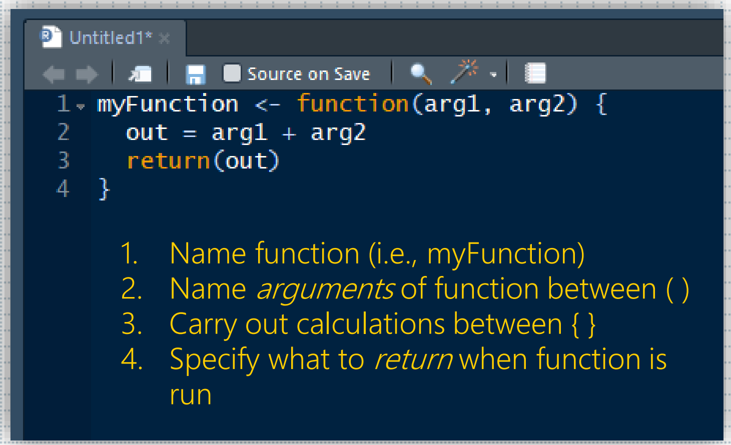
Learn basic programming concepts

- THREE MUST-LEARN PROGRAMMING CONCEPTS
  - FUNCTIONS
    - » *EFFICIENCY* → PERFORM CALCULATIONS AND RETURN OUTPUT
    - » USEFUL FOR RUNNING FREQUENTLY USED/REQUIRED PROCEDURES
      1. WRITING FUNCTIONS NOT STRICTLY NECESSARY…BUT ARE POWERFUL!

1 IF-ELSE

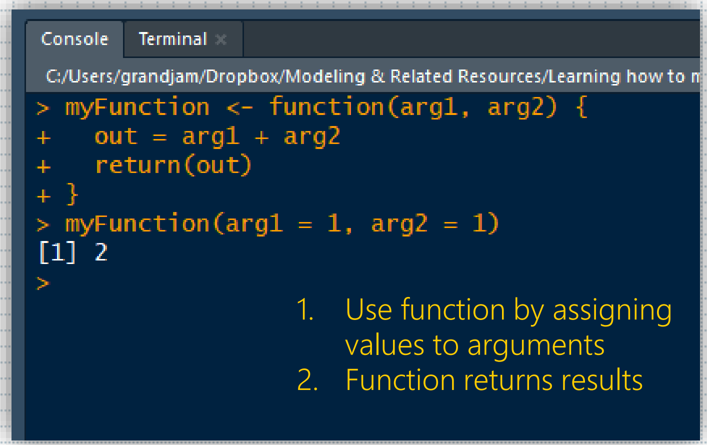
2 LOOPS

3 FUNCTIONS



```
myFunction <- function(arg1, arg2) {
  out = arg1 + arg2
  return(out)
}
```

1. Name function (i.e., myFunction)
2. Name *arguments* of function between ( )
3. Carry out calculations between { }
4. Specify what to *return* when function is run



```
C:/Users/grandjam/Dropbox/Modeling & Related Resources/Learning how to m
> myFunction <- function(arg1, arg2) {
+   out = arg1 + arg2
+   return(out)
+ }
> myFunction(arg1 = 1, arg2 = 1)
[1] 2
>
```

1. Use function by assigning values to arguments
2. Function returns results

PROGRAMMING FUNDAMENTALS 101

Step 2

Learn basic programming concepts

- My advice for the first-time modeler/programmer
  - Break the problem down
    - What do you want accomplished?
    - What steps are involved?
    - What should the result of each step look like?
  - Fail fast by testing often
    - Run new code frequently
    - confirm that what should happen does happen
  - Comment EVERYTHING
    - Code doesn't run slower
    - You & others will know what is happening

Programming Fundamentals 101

Conclusion & Where to Go From Here

- Powerful approach for thinking about organizational, social, and psychological phenomena
  - Tool for representing dynamics, process, and *what happens*
- Necessitates a way of thinking and skillset that our science is <u>not</u> well versed in → ***massive opportunity!***
- Next steps
  - Read more about models and modeling
  - Look at, run, and try to code some simple models

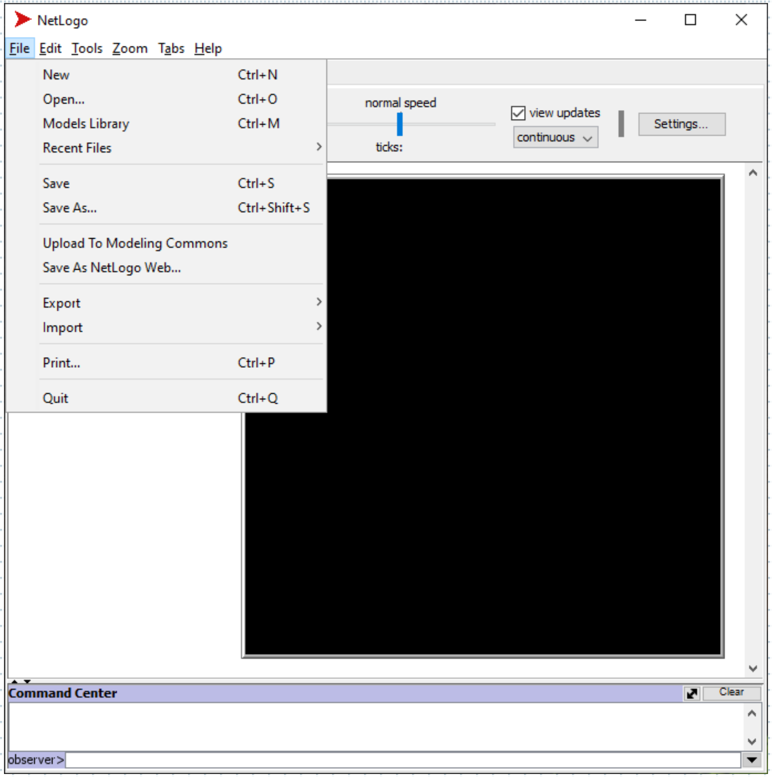
Step 1 — Train yourself to think in terms of actors, not factors

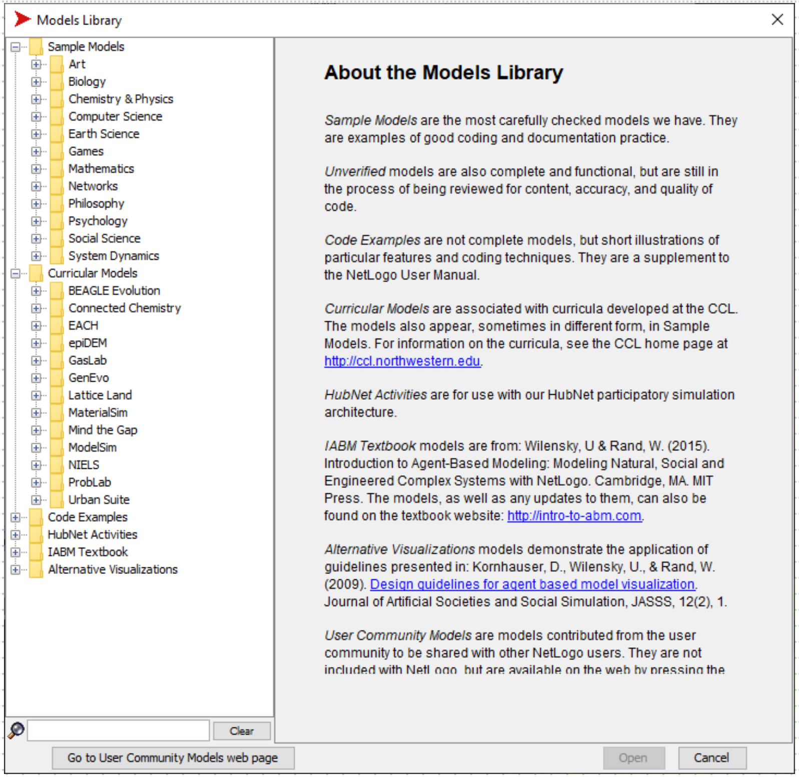
Step 2 — Learn basic programming concepts

- FURTHER READING ON COMPUTATIONAL MODELING (FOR THE ORG SCIENCES)
  - Journal articles
    - Davis, J.P., Eisenhardt, K.M., & Bingham, C.B. (2007). Developing theory through simulation methods. *Academy of Management Review, 32*, 480-499.
    - Harrison, J.R., Lin, Z., Carroll, G.R., & Carley, K.M. (2007). Simulation modeling in organizational and management research. *Academy of Management Review, 32*, 1229-1245.
    - Kozlowski, S.W.J., Chao, G.T., Grand, J.A., Braun, M.T., & Kuljanin, G. (2013). Advancing multilevel research design: Capturing the dynamics of emergence. *Organizational Research Methods, 16*, 581-615.
    - Vancouver, J.B., & Weinhardt, J.M., (2012). Modeling the mind and the milieu: Computational modeling for micro-level organizational researchers. *Organizational Research Methods, 15*, 602-623.
    - Weinhardt, J.M., & Vancouver, J.B. (2012). Computational models and organizational psychology: Opportunities abound. *Organizational Psychology Review, 2*, 267-292.
  - Books
    - Sterman, J.D. (2000). *Business dynamics*. Irwin/McGraw Hill.
    - Wilensky, U., & Rand, W. (2015). *An introduction to agent-based modeling: Modeling natural, social, and engineered complex systems with NetLogo*. MIT Press.

CONCLUSION & WHERE TO GO FROM HERE

- LEARNING TO CODE MODELS
  - NetLogo (https://ccl.northwestern.edu/netlogo/download.shtml)
  - File → Models Library





About the Models Library

Sample Models are the most carefully checked models we have. They are examples of good coding and documentation practice.

Unverified models are also complete and functional, but are still in the process of being reviewed for content, accuracy, and quality of code.

Code Examples are not complete models, but short illustrations of particular features and coding techniques. They are a supplement to the NetLogo User Manual.

Curricular Models are associated with curricula developed at the CCL. The models also appear, sometimes in different form, in Sample Models. For information on the curricula, see the CCL home page at http://ccl.northwestern.edu.

HubNet Activities are for use with our HubNet participatory simulation architecture.

IABM Textbook models are from: Wilensky, U & Rand, W. (2015). Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo. Cambridge, MA. MIT Press. The models, as well as any updates to them, can also be found on the textbook website: http://intro-to-abm.com.

Alternative Visualizations models demonstrate the application of guidelines presented in: Kornhauser, D., Wilensky, U., & Rand, W. (2009). Design guidelines for agent based model visualization. Journal of Artificial Societies and Social Simulation, JASSS, 12(2), 1.

User Community Models are models contributed from the user community to be shared with other NetLogo users. They are not included with NetLogo, but are available on the web by pressing the

CONCLUSION & WHERE TO GO FROM HERE

- LEARNING TO CODE MODELS
  - Start by replicating simple models
    - Don't worry if you're not "interested" in the topic per se…the point is to practice fundamentals!

Martell, R.F., Lane, D.M., & Emrich, C. (1996). Male-female differences: A computer simulation. *American Psychologist, 51*, 157-158.

Why are women underrepresented in senior leadership positions?

Scullen, S.E., Bergey, P.K., Aiman-Smith, L. (2005). Forced distribution rating systems and the improvement of workforce potential: A baseline simulation. *Personnel Psychology, 58*, 1-32.

How does the quality of a selection system impact an organization's performance potential?

  - Full R code for both models can be downloaded from my GitHub

https://github.com/grandjam/SIOP2021Models

CONCLUSION & WHERE TO GO FROM HERE

https://github.com/grandjam/SIOP2021Models

Thank You!

James A. Grand

E-mail: grandjam@umd.edu

University of Maryland