

# Introduction to $\mathcal{R}$

## Session 3: Data management

Dag Tanneberg<sup>1</sup>

Potsdam Center for Quantitative Research  
University of Potsdam, Germany  
October 11/12, 2018

---

<sup>1</sup>Chair of Comparative Politics, UP, [dag.tanneberg@uni-potsdam.de](mailto:dag.tanneberg@uni-potsdam.de)

# Introduction

# Outline

- 1 Introduction
- 2 Sequential Execution
- 3 Iterative Execution
- 4 Conditional Execution
- 5 Summary
- 6 Practical Challenges

# Sequential Execution

# Sequential Execution

- Run a script
  - step by step;
  - from start to end.

```
A <- "Well"
B <- "hello there."
paste(A, B, sep = ", ")
rm(A, B, C) # Explain the error.
```

# Iterative Execution

# A.K.A. Looping

- Execute statement(s) repeatedly
  - a. over a set of values
  - b. as long as some condition holds
  - c. until an abort condition is met
- Includes: for, while, and repeat
- Typical use-case: transform several variables

## for()-Loops<sup>2</sup>

- repeats statements for each element on an input set

```
# Generic example
```

```
for (VALUE in THAT) { # Do THIS for each VALUE in THAT  
  THIS  
}
```

```
# A first working example
```

```
for (value in c("Waiting", "for", "statistics.")) {  
  print(value)  
}
```

- for() creates an object called VALUE
- reassigns VALUE for each element in the set THIS

---

<sup>2</sup>People don't like for(). For alternatives see <https://bit.ly/2IEbeGj>.



# for()-Loops, contd.

- for() returns nothing unless told to<sup>3</sup>
- Save the output to an object
- Good practice:
  - Execute on a set of integers
  - Index both object and storage simultaneously

```
words <- c("So", "how's", "looping", "so", "far?")  
chr <- vector("character", length = length(words))  
for (i in 1:length(words)){  
  chr[i] <- words[i]  
}
```

---

<sup>3</sup>“for loops are like Las Vegas: what happens in a for loop stays in a for loop”  
(Gorrelmund 2014: 164).

## Quick Exercise

Remember last session's data management challenge? Let's try to express our solution as a `for()`-Loop.

```
grade_quantiles <- quantile(  
  student_data[, "grade"], probs = c(.2, .4, .6, .8)  
)  
student_data[, "grade_alp"] <- "F"  
student_data[  
  student_data[, "grade"] > grade_quantiles["20%"],  
  "grade_alp"  
] <- "D"  
# ... and so on until A.
```

# While()-Loops

- Rerun statement(s) as long as some condition is TRUE
- Condition should be a logical test
- Remember “Groundhog Day”
  - Include a change of condition in the while()'s body!

```
k <- 0
while (k < 20) {
  k <- k + 1
  print("Still running")
}
```

- Returns nothing unless told to

# Repeat()-Statements

- Reruns statement(s) until meets **break**

```
chr <- "All work and no play makes Jack a dull boy"
k <- 0
repeat {
  print(chr)
  k <- k + 1
  if (k > 100) break
}
```

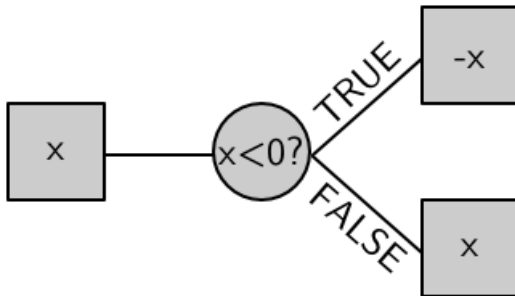
But...

How do we tell  $\mathcal{R}$  to execute some code conditionally?

## Conditional Execution

# Intuition

- How does the absolute value  $|x|$  algorithmically work?



- *Different* operations follow depending on some condition
- $\rightarrow$  code handles parallel cases

# if()-Statements

- Code executes if and only if some condition is TRUE
- Condition **must** evaluate to a single TRUE/FALSE statement

```
if (THIS) { # If this is TRUE  
  THAT # then do THAT.  
}  
  
x <- 4  
if (x < 0) {  
  x <- -1 * x  
}
```

# if()-Statements: What will this return?

```
# Example 1 =====  
x <- 1  
if (TRUE) {  
  x <- 2  
}  
  
# Example 2 =====  
x <- 1  
if (x == 1) {  
  x <- 2  
  if (x == 1) {  
    x <- 3  
  }  
}
```



# else()-Statements

- tell  $\mathcal{R}$  what to do should if() evaluate to FALSE
- multiple if/else statements can be nested

```
if (this) {  
  Plan A  
} else {  
  Plan B  
}  
  
dec <- 3.141 # Example: Round a decimal to integer  
if (dec - trunc(dec) >= 0.5) {  
  dec <- trunc(dec) + 1  
} else {  
  dec <- trunc(dec)  
}
```

# Summary

# What was that about?

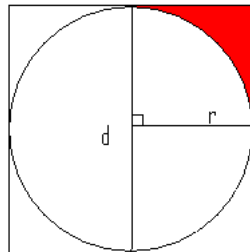
- Code can be executed sequentially, iteratively, or conditionally
- Sequential execution is the norm
- Iterative execution runs the same code repeatedly
  - **for()** reruns statement(s) for all members of a set
  - **while()** reruns statement(s) as long as a condition is met
  - **repeat()** reruns statement(s) until it encounters **break**
- Conditional execution manages parallel cases
  - **if()** runs statement(s) if a condition evaluates to TRUE
  - **else()** runs statement(s) if that same condition is FALSE

# Practical Challenges

# In a world where humanity forgot the value of $\pi$ ...

... we will uplift civilization by Monte Carlo simulation.

Set up a simulation which allows you to generate an estimate of  $\pi$  from the chance to hit a circle perfectly inscribed in a square with a randomly thrown dart.



$$p(Hit) = \frac{A_{ci}}{A_{sq}} = \frac{\pi r^2}{(2r)^2}$$
$$\pi = 4p(Hit)$$

# Understanding Collider Bias

In causal inference, collider bias results when we condition on a variable that is causally influenced by one or more other variables. The effect will be a spurious correlation between the collider's ancestors. Setup a simulation to demonstrate collider bias.

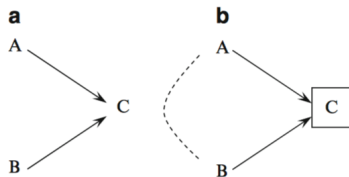


Figure 1: Collider Bias Visualized<sup>4</sup>

---

<sup>4</sup>Ellwert, F. 2013. Graphical Causal Models. In: S.L. Morgan (ed.), Handbook of Causal Analysis for Social Research, p. 251