Introduction to $\mathcal{R}$

Session 3. Basic Program Flow

Dag Tanneberg[1]

Potsdam Center for Quantitative Research
University of Potsdam, Germany
November 8/9, 2019

[1] Quantitative Social Scientist, www.dtanneberg.de

Introduction

Outline

Sequential Execution

Sequential Execution

- Run a script
  - step by step;
  - from start to end (or other condition).

```
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[2]

- 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 THAT

[2] People don't like for(). For alternatives see https://bit.ly/2IEbeGj.

for()-Loops, contd.

- for() returns nothing unless told to[3]
- 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]
}
```

[3] "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()-Statements

- Rerun statement(s) as long as some condition is TRUE
- Remember "Groundhog Day"?

```
k <- 0
while (k < 20) {  # statement evaluates to ONE value
  cat(k, "and counting...", "\n")
  k <- k + 1  # Make sure that your condition fails!
}
```

repeat()-Statements

- Rerun statement(s) until **break** statement is met

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

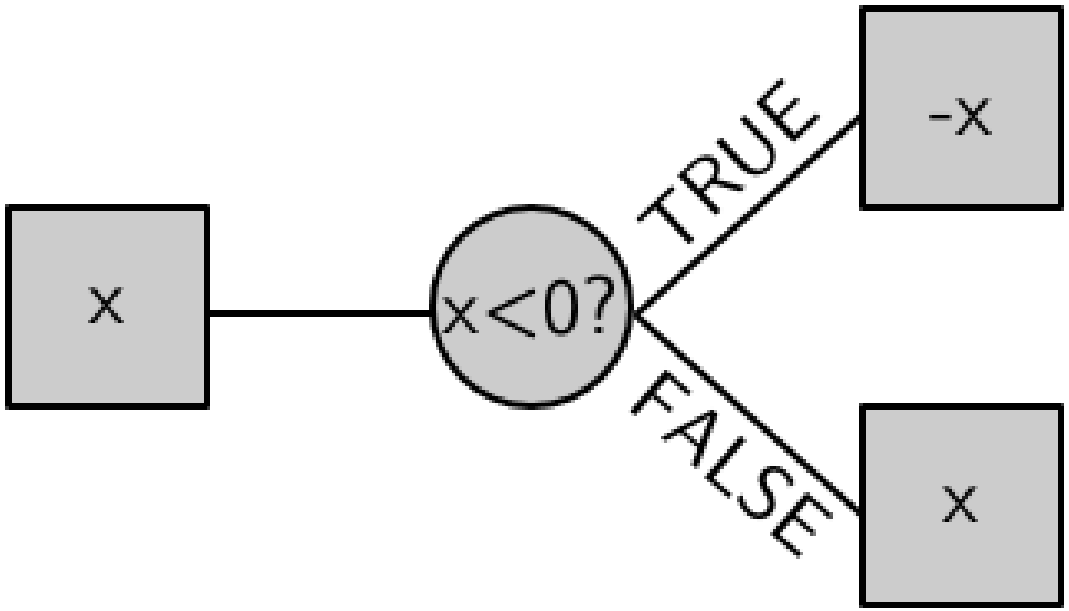
But...

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

Conditional Execution

Intuition

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



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

Detour: Logical Tests in *R*

- Check the truth value of some condition
- All tests return a logical values
- Obey element-wise execution and recycling

Operator	Syntax	Test
$>$	a $>$ b	Is a greater than b?
$>=$	a $>=$ b	Is a at least equal to b?
$<$	a $<$ b	Is a less than b?
$<=$	a $<=$ b	Is a at most equal to b?
$==$	a $==$ b	Is a equal to b?
$!=$	a $!=$ b	Is a not equal to b?
%in%	a %in% c(a, b, c)	Is a in the set c(a, b, c)?

Longer Detour: Boolean Operators in *R*

- Operate on several logical tests
- *Usually* return a single logical value

Operator	Syntax	Test
&	cond1 & cond2	Are both conditions true?
|	cond1 & cond2	Is at least one condition true?
xor	xor(cond1, cond2)	Is either cond1 or cond2 true?
!	!cond1	Negate cond1.
any()	any(cond1, cond2)	Is any condition true?
all()	all(cond1, cond2)	Are all conditions true?
%in%	a %in% c(a, b, c)	Is a in the set c(a, b, c)?

if()-Statements

- Code executes if and only if some condition is TRUE
- Condition should 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
}
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 <- pi # Example: Round a decimal to integer
if (dec - trunc(dec) >= 0.5) {
  dec <- trunc(dec) + 1
} else {
  dec <- trunc(dec)
}
```

But I want to operate on an entire vector!

- Looping: Use vectorized functions, e.g. apply(), sapply(), etc.
- Conditional execution: Use `ifelse(<test>, <yes>, <no>)`

```
x <- 1:6
evens <- ifelse(x %% 2 == 0, 1, 0)
rbind(x, evens)
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6]
## x         1    2    3    4    5    6
## evens     0    1    0    1    0    1
```

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 statesment(s) if a condition evaluates to TRUE
  - **else()** runs statesment(s) if that same condition is FALSE
  - **ifelse()** is a vectorized version

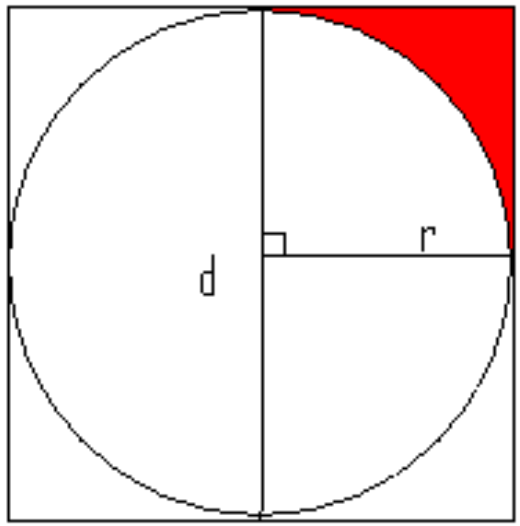
Practical Challenge

In a world where humanity forgot the value of π ...

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

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

You can do this in four lines!



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

$$\pi = 4p(Hit)$$