University of Zurich[UZH]

Department of Geography

GEO 812
Getting started with R for Spatial Analysis

Session 3: Programming in R

Peter Ranacher

September 2019

Learning objectives for Session 3

You are able to

- explain loops and understand when not to use them in R
- use conditional statements
- work with functionals
- create your own functions, test and debug them

A simple task:

Compute the mean for every column in `msleep`, but only if the column contains numeric data.

The very, very naïve approach:

`colnames(msleep)`	get all column names
`is.numeric(msleep$name)`	first column is not numeric
`is.numeric(msleep$genus)`	second column is not numeric
…	
`is.numeric(msleep$sleep_total)`	TRUE!! We found a numeric column.
`mean(msleep$sleep_total)`	Compute the mean
…	
`is.numeric(msleep$bodywt)`	We found the last numeric column.
`mean(msleep$bodywt)`	Compute the mean

Reduce copy-pasting in your code whenever possible

- easier to see the intent of your code
- easier to respond to changing requirements → only change code in one place
- fewer bugs, since each line of code is used in more than one place

Rule of thumb: Never copy-and-paste more than twice!

Rewrite the task in a more generic way

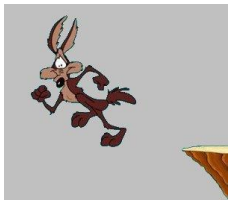
FUNCTION

1. Go through all columns in the tibble. LOOP

2. If the element is numeric, compute the mean. CONDITIONAL STATEMENT
   If not, don't do anything.

What is 1?
What is 2?
What are 1 and 2 combined?

Conditional statements: if



→

IF the **CONDITION** is **TRUE** → **CONSEQUENCE**

CONDITION: a logical expression that is either TRUE or FALSE

CONSEQUENCE: code executed when condition is TRUE

Example:

```
x <- -7
if (x < 0) {cat(x, "is negative")}
```

Q: What is `cat()` ?
A: Concatenate and print.

Conditional statements: else

IF the **CONDITION** is **TRUE** → **CONSEQUENCE**

ELSE → **ALTERNATIVE CONSEQUENCE**

ALTERNATIVE CONSEQUENCE: code executed when condition is FALSE

Example:

```
x <- 8
if (x < 0) {
  cat(x, "is negative")
} else if (x == 0)  {
  cat(x, "is null")
} else {
  cat(x, "is positive")
}
```

What does `else if` do?

Operators for comparison

Operator for comparison	description
==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

Always use | | and & & to combine logical expressions and never | and &

```
x <- 0.6
if (x > 0 && x < 4) {
  cat(x, "is positive and smaller than four.")
}
```

Functions for comparison

Function	description
`any (x)`	Given a set of logical vectors is at least one of the values true?
`all (x)`	Given a set of logical vectors, are all of the values true?

`is.numeric(x)`
`is.character(x)`
…
	TRUE if x is of type numeric / character

Example:

```
x <- c(-1, -3, -6, 8, 9,5)
if (any(x > 0)) {
  cat("Some numbers in ", x, " are positive.")
}
```

Loops

- `for` loops

  run a code block a certain number of times, e.g. for each column in a tibble

- `while` or conditional loops

  run a code block until a certain condition is met

`for` loops

Example: a loop to calculate the cube of numbers 1 to 5

```
x <- 1:5
output <- vector("double", length(x))

for (i in 1:5){
    i_cubed <- i * i * i
    output[[i]] <- i_cubed
}
```

output

sequence

body

- **Output:**
  empty vector to allocate space for output. Important for efficiency!
- **Sequence:**
  determines what to loop over
- **Body:**
  code that it is repeated

`while` loops

while – condition is evaluated at the beginning

repeat – condition is evaluated at the end (loop is at least entered once)

Example: square all numbers smaller than 100

```
i <- 1
while (i < 100) {
    i_squared <- i * i
    output[[i]] <- i_squared
    i <- i + 1
}
```

condition

body

Why is the output not defined before entering the loop?

To loop or not loop?

- loops offer a good view on what is supposed to happen
- require an understanding of the data and the process you want to carry out, BUT

KNOW your loops and try to GET RID of them whenever possible!

Two approaches to get rid of loops:

- Vectorization
- Functions and functionals (`apply` family)

Vectorization

Add vectors A and B:

```
A <- c(1, 2, 4, 1)
B <- c(2, 1, 5, 1)
```

Loop over elements of vector:

NO!

NO!

```
C <- vector("double", length(A))
for (i in 1:length(A)){
  C[i] <-A[i] + B[i]
}
```

NO!

NO!

NO!

Vectorization:

```
C <- A + B
```

YES!

repeated operations on simple numbers
→ single operations on vectors

Functions to get rid of loop

What was the average age of passengers on the Titanic?

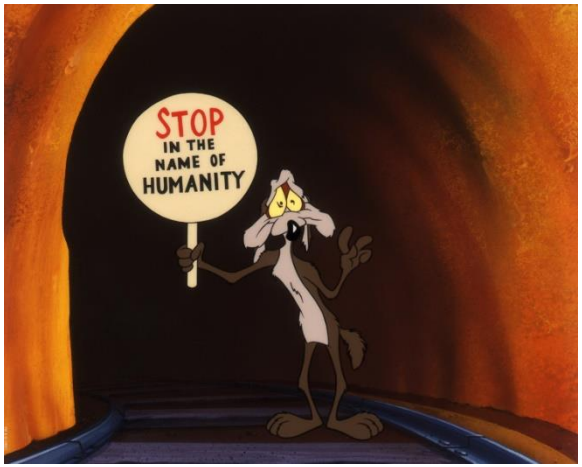
NO!

```
summed_age <- 0
n <- length(na.omit(titanic_survival$age))
for (i in titanic_survival$age){
  if (!is.na(i)) {
    summed_age <- i + summed_age}
}
summed_age / n
```

NO!

NO!

NO!



Any time you think you have to do a loop in R….

… look if there is a function that can do the same operation WITHOUT a loop!

```
mean(titanic_survival$age, na.rm = TRUE)
```

YES!

Functionals

function that takes a **function** as an input and returns a vector as output

`lapply()`

- takes a vector and a function as input
- calls the function for each element of the vector
- returns a list as a result

Example: `lapply()` over list

```
data_list <- list(msleep, titanic_survival)
lapply(X = data_list, FUN = is.data.frame)
```

Example: `lapply()` over columns of a data frame

```
lapply(msleep, is.character)
```

Run `sapply(msleep, is.character)`
How does it differ from the `lapply()` result?

Functionals continued

`apply()`

- takes a matrix and a function as input
- calls the function for each row (`MARGIN=1`) or column (`MARGIN=2`) of the matrix
- returns a vector

```
apply(X = msleep, MARGIN = 2, FUN = max, na.rm = TRUE)
```

Check the type of the output!
Why is it "character"?

Exercise 5

1. Run the following code for `n = m = 10` and for `n = m = 10000`. Explain what you observe!

```
A <- matrix(data = rnorm(n * m), nrow = n, ncol = m)
## Vectorization
system.time(A^2)
## Loop
system.time(for (i in 1:n){
          for (j in 1:m){
            A[i, j] <- A[i, j]^2
            }})
```

`rnorm(x)` generates x random numbers that follow a normal distribution with mean = 0 and standard deviation = 1.

2. Use functions from the `apply` family to

- get all numeric columns of `msleep` (hint: use `sapply()`)
- compute the mean for all numeric columns (hint using indexing first!)

Writing your own functions

A function to calculate the cube root of a number

function name

argument(s)

```
cube_root <- function(nr_to_cube)
  {
    result <- nr_to_cube ^ (1/3)
    return(result)
  }
```

What does the function do?

What does the function return?

Call!

```
cube_root(1000)
```

There is a problem with the function we've just defined….

```
cube_root(1000)
```
OK

```
cube_root(-1000)
```
NaN (^ only works for positive bases)

```
cube_root("Busta Rhymes")
```

Data checking

Make sure that the data given to the function are of the right type!

Check if input is numeric

```
cube_root <- function (x){
  if (!is.numeric(x)) {stop("x must be a
number")}
  else {
      if (x >= 0) {result <- x ^ (1/3)}
      else {result <- -(-x)^(1/3)}
    return (result)}}
```

Check if input is > 0

Debugging

What to do when a function you wrote

- does not work
- produces the wrong results

Have a look at the function!

```
debug(cube_root)
cube_root("Busta Rhymes")
```

Enter a variable: show the value of the variable

Stop entering the debug mode

```
undebug(cube_root)
```

Read the data collected in class

```
stats_geo812 <- read.csv(file = "data/stats_geo812.csv",
                         header = T, sep = ",")
```

Exercise 6

1. Write a function `d_great_circle` to compute the great-circle distance (d) between two points on the Earth surface. This is the formula for d:

$$\mathrm{d} = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

   It takes as input the latitude (φ) and longitude (λ) of two locations on the Earth surface and the Earth radius r. You can set r to 6371 km.

   - Perform data checking (is the input numeric, is it a valid latitude/longitude?)
   - In R, `cos()` and `sin()` take radians as input! This function helps you with the conversion:
     `deg2rad <- function(deg) {(deg * pi) / (180)}`

2. Compute the distance from Zurich (φ = 47.3686498, λ = 8.5391825) to your holiday locations (`holidays_lat`, `holidays_lon`) in `stats_geo812`.
   - Use `mutate()` rather than `apply()`. Why?

Learning objectives revisited

You are able to

- explain loops and understand when not to use them in R
- use conditional statements
- work with functionals
- create your own functions, test and debug them