QTM 150

Week 13 – Functions, Loops, and Other Programming

Umberto Mignozzetti
Apr 23

Recap

You now know:

- The main objects in R.
- How to do basic operations with datasets.
- How to create graphs and plots.
- Data manipulation with `dplyr`.
- Graphs with `ggplot`.

Great job!!

Do you have any questions?

Today we are going to learn functions, loops, and elements of statistical programming!

This week

We will have a **quiz** posted today after 4:00 PM. Due by **Monday**.

We will **not** have a **problem set** this week. Focus on your final projects.

Our GitHub page is: https://github.com/umbertomig/qtm150

Final Project

The final projects are due by Apr 28 (Wednesday).

Next class we will have our final project presentations. Make sure to come to class.

Next class is the only class that I require attendance synchronously. Sorry for the folks in different time zones, but this one is important.

Make sure to upload your slides before Friday. Slides are due by Apr 29. I'll this deadline to Canvas later today.

Today's Agenda

`if` statements

`function` creation

`for` loops

Getting Started

Getting Started: loading packages

```r
# Loading tidyverse
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyv

## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.1.0     ✓ dplyr   1.0.5
## ✓ tidyr   1.1.3     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1

## ── Conflicts ────────────────────────────────────────── tidyverse_c
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Loading datasets

```
# Loading tips dataset
tips ← read.csv('https://raw.githubusercontent.com/umbertomig/qtm
head(tips, 2)
```

```
##   obs totbill  tip sex smoker day  time size
## 1   1   16.99 1.01   F     No Sun Night    2
## 2   2   10.34 1.66   M     No Sun Night    3
```

```
# Loading PErisk dataset
PErisk ← read.csv('https://raw.githubusercontent.com/umbertomig/
head(PErisk, 2)
```

```
##     country courts      barb2 prsexp2 prscorr2    gdpw2
## 1 Argentina      0 -0.7207754       1        3  9.69017
## 2 Australia      1 -6.9077550       5        4 10.30484
```

```
data(USArrests)
```

if statements

if statements

In statements control the flow of a code.

They create a condition based on some variable, and execute a piece of code when the condition is `TRUE`.

Example in pseudo-code:

```
if (number > 10) {
   ... execute code for the bigger than 10 case ...
} else {
   ... execute code for the smaller than or equals 10 case ...
}
```

ifelse

In R we have a function to conveniently build if-else statements.

If-else consist in binary conditions: do something if true, or something else if false.

Syntax: `ifelse(condition, code-if-true, code-if-false)`

For example:

```
n = 4
ifelse(n>2, n^2, n/5)

n = 1
ifelse(n>2, n^2, n/5)

n = 2
ifelse(n>2, n^2, n/5)
```

ifelse

More examples:

```r
## no need to interact with the condition:
n = 2
ifelse(n>2, 'bigger than 2', 'smaller than or equal 2')

## gender: vector operation
gen <- c(0,0,0,1,1,0,1,1,1,0)
ifelse(gen == 1, 'Fem', 'Masc')

## gender: vector operation
gen <- c(0,0,2,1,1,2,1,1,1,0)
ifelse(gen == 1, 'F', ifelse(gen == 0, 'M', 'Non-binary'))
```

Your turn: in the tips dataset, take the variable `day`, and create a condition that shows whether the day is weekday or weekend.

"if - else if - else" statements

There are statements that are bigger in nature than simple ifelse ones.

Example in pseudo-code:

```
if (number > 10) {
   ... execute code for the bigger than 10 case ...
} else if (number > 2 & number ⩽ 10) {
   ... execute code for this condition ...
} else {
   ... execute code for otherwise ...
}
```

"if - else if - else" statements

Example:

```r
n = 20
if(n==1) {
  print('Number equals 1')
} else if (n==2) {
  print('Number equals 2')
} else if (n==3) {
  print('Number equals 3')
} else if (n==4) {
  print('Number equals 4')
} else if (n<0) {
  print('Number smaller than zero')
} else {
  print('Invalid number!')
}
```

```
## [1] "Invalid number!"
```

functions

functions

Sometimes we need to create our own functions.

Create functions is simple. The syntax is straightforward:

```
name_function ← function(par1, par2, par3, ...) {
   ... code execute ...
  return( ... return smt ... )
}
```

Example in pseudo-code:

```
testn ← function(n) {
  if (number > 10) {
    ... execute code for the bigger than 10 case ...
  } else {
    ... execute code for the smaller than or equals 10 case ...
  }
  return( ... smt ... )
```

functions

.font130[

Example: BMI index function

```r
bmi <- function(w, h) {
  bmi = w / (h^2)
  return(bmi)
}
# BMI person 70 kg and 1.75 m
bmi(70, 1.75)
```

```
## [1] 22.85714
```

Your turn create a function that receives weight in pounds and heights in inches and return the BMI.

loops

loops

There are two ways of creating loops: `while` and `for`.

While loops are useful to run a piece of code until a condition is satisfied. For example:

```
i = 1
while(i < 50) {
  print(i)
  print('i is smaller than 50!')
  print(i^10)
  i = i+1
}
```

loops

Example: infinite loop:

```
while(TRUE) {
  x = readline("Type something or break: ")
  if(x == 'break'){
    break
  }
  cat("You typed: ", x)
}
```

loops

The `for` loop is a bit different: it receives a counter, and run until the counter is not exhausted.

Example:

```
for (i in 1:10) {
  print(i)
  print(i^5)
}
```

loops

Example: let's run over the columns of a dataset, and display the class of each variable:

```
for (i in names(tips)) {
  print(class(tips[,i]))
}
```

Your turn: Create a function that displays the summary of the numeric or integer variables, and table of the other variables.

Questions?

Have a great weekend!