Problem Set 2 Solution

Andreas Bender, Philipp Kopper, Philip Studener

31 October 2023

Ressources

1) Read chapter 4 on lists in "The Art of R Programming". You can ignore "Extended Examples".
2) Optionally, complete the Datacamp Units provided on the Moodle page for this topic.

Keyboard Shortcuts

Something that can greatly increase your efficiency when programming is the use of keyboard shortcuts. You may already know simple shortcuts, such as `CTRL + C` and `CTRL + V` (`CMD + C` and `CMD + V` on Mac) for copying and pasting text. Try incorporating the following shortcuts into your workflow for this week's exercise sheet:

Shortcut Windows	Shortcut Mac	What does it do?
`CTRL + Enter`	`CMD + Enter`	Run Code in the line where your cursor is
`CTRL` + ←/→	`OPT` + ←/→	Skip words with cursor
`CTRL + Shift` + ←/→	`OPT + Shift` ←/→	Mark the left/right word
`CTRL + Shift + Alt + M`	`CMD + Shift + OPT + M`	Rename the marked variable in scope

Exercises

1. Create the following list, that stores information on three students (`name`, `gender`) and the points they reached in 3 partial exams: `points_ex1` (max. 20 points), `points_ex2` (max. 20 points) and `points_ex3` (max. 100 points). Name the object `mylist`.

```
mylist <- list(
  name       = c("Linda", "Eva", "Matthias"),
  gender     = c("F", "F", "M"),
  points_ex1 = c(15, 18, 12),
  points_ex2 = c(9, 16, 8),
  points_ex3 = c(83, 95, 39))
mylist
```

```
## $name
## [1] "Linda"    "Eva"      "Matthias"
##
## $gender
## [1] "F" "F" "M"
##
## $points_ex1
## [1] 15 18 12
##
## $points_ex2
## [1]  9 16  8
##
## $points_ex3
## [1] 83 95 39
```

a. Extract the variable `points_ex1` from `mylist` in at least two different ways (*Hint: ARP, Section 4.2.1*). Discuss differences/advantages of the different approaches.

```
mylist[["points_ex1"]] # Outputs the vector
```

```
## [1] 15 18 12
```

```
mylist["points_ex1"] # Outputs a list with only points_ex1 in it
```

```
## $points_ex1
## [1] 15 18 12
```

```
mylist$points_ex1 # Outputs the vector
```

```
## [1] 15 18 12
```

```
mylist[[3]] # Outputs the vector.
```

```
## [1] 15 18 12
```

```
# Is bad practice, because what if the order of the list changed?
mylist[3] # Outputs a list.
```

```
## $points_ex1
## [1] 15 18 12
```

```
# Is bad practice, because what if the order of the list changed?
```

b. Output the names (also called "tags") of all variables (also called "components") in the list (*Hint: ARP, Section 4.3*).

```
names(mylist)
```

```
## [1] "name"       "gender"     "points_ex1" "points_ex2" "points_ex3"
```

c. Add a new element `passed` to the list that contains the information if someone passed the exam or not (`TRUE` or `FALSE`); *Hint: ARP, Section 4.2.2.* The exam is passed, if at least 50% of the total point maximum have been reached. To achieve this goal you can create an intermediate variable `total_points` that indicates how many points each student reached over the 3 partial exams and delete it afterwards (*Hint: The number of points reached by each person is the sum of all points from exams 1 through 3.*). The resulting list is given below:

```
threshold_passed <- (20 + 20 + 100) / 2 # 50% of total achievable points
mylist$points_total <- mylist$points_ex1 + mylist$points_ex2 + mylist$points_ex3
mylist$passed <- mylist$points_total >= threshold_passed
mylist$points_total <- NULL
mylist
```

```
## $name
## [1] "Linda"    "Eva"      "Matthias"
##
## $gender
## [1] "F" "F" "M"
##
## $points_ex1
## [1] 15 18 12
##
## $points_ex2
## [1]  9 16  8
##
## $points_ex3
## [1] 83 95 39
##
## $passed
## [1]  TRUE  TRUE FALSE
```

d. Create another variable `passed_factor`, that is a factor variable with values `"yes"`, if variable passed has value `TRUE` and `"no"` otherwise. Confirm that variable `passed_factor` has class `factor`. The resulting list is given below:

```
mylist$passed_factor <- factor(mylist$passed, levels = c(TRUE, FALSE),
  labels = c("yes", "no"))
mylist
```

```
## $name
## [1] "Linda"    "Eva"      "Matthias"
##
## $gender
## [1] "F" "F" "M"
##
## $points_ex1
## [1] 15 18 12
##
## $points_ex2
## [1]  9 16  8
##
## $points_ex3
```

```
## [1] 83 95 39
##
## $passed
## [1]  TRUE  TRUE FALSE
##
## $passed_factor
## [1] yes yes no
## Levels: yes no
```

```
class(mylist$passed_factor)
```

```
## [1] "factor"
```

e. Order the list such that the element `passed` is the first element in `mylist` (*Hint: Make sure your solution works, even if the position of variables in the list is changed; see* `?which`). The result is given below:

```
# Hard coded without the Hint in the question.
# It shows the idea behind reordering a list.
# Try to avoid this if possible though:
# mylist[c(7, 1, 2, 3, 4, 5, 6)]

# Better version.
ind <- 1:length(mylist)
ind_passed <- which(names(mylist) == "passed")
mylist <- mylist[c(ind_passed, ind[-ind_passed])]
mylist
```

```
## $passed
## [1]  TRUE  TRUE FALSE
##
## $name
## [1] "Linda"    "Eva"      "Matthias"
##
## $gender
## [1] "F" "F" "M"
##
## $points_ex1
## [1] 15 18 12
##
## $points_ex2
## [1]  9 16  8
##
## $points_ex3
## [1] 83 95 39
##
## $passed_factor
## [1] yes yes no
## Levels: yes no
```

f. Sometimes names in data analysis can be non-compliant with data privacy issues. Thus, delete the variable `name` from `mylist`.

```
mylist$name <- NULL
mylist
```

```
## $passed
## [1]  TRUE  TRUE FALSE
##
## $gender
## [1] "F" "F" "M"
##
## $points_ex1
## [1] 15 18 12
##
## $points_ex2
## [1]  9 16  8
##
## $points_ex3
## [1] 83 95 39
##
## $passed_factor
## [1] yes yes no
## Levels: yes no
```

g. Of what type are the two objects below? Why?

```
mylist[1]
```

```
## $passed
## [1]  TRUE  TRUE FALSE
```

```
mylist[[1]]
```

```
## [1]  TRUE  TRUE FALSE
```

```
class(mylist[1]) # list
```

```
## [1] "list"
```

```
typeof(mylist[1])
```

```
## [1] "list"
```

```
class(mylist[[1]]) # logical
```

```
## [1] "logical"
```

```
typeof(mylist[[1]])
```

```
## [1] "logical"
```

```
# mylist[1] outputs a list with the first element of mylist,
# so it returns a sublist of the original list
# mylist[[1]] outputs the 1st component of mylist and has
# the type or class of that component
```

Bonus question: Change the name of the object `mylist` from `mylist` to something that reflects the information contained within the object. Use a suitable keyboard shortcut to change all occurrences of `mylist` in your `R` file at once. Then re-run your code. Can you still access `mylist`? Why?

Solution: The shortcut in question was `CTRL + Alt + Shift + M` / `CMD + OPT + Shift + M`. To use it you mark one instance of `mylist` then press it. Your cursor should now be at every instance of `mylist` and you can change the name simultaneously. Yes, since you already defined `mylist` and it got saved in your environment you can still access it. Best practice is to remove variables from your environment if you don't use them anymore (`rm(mylist)`).

2. Hobby-Statistician AB is teaching three small cohorts of first semester students. He collects their grades from their mid-term exam in a `list` in `R`. He named the cohorts with respect to the day that he taught them.

```
cohorts <- list(
  Monday    = c(1.0, 1.0, 5.0, 3.3, 2.0),
  Tuesday   = c(2.0, 2.0, 4.0, 5.0, 1.0, 1.3, 1.7, 2.0),
  Wednesday = c(3.3, 4.0, 5.0, 1.0, 5.0, 5.0))
```

In this exercise we are going to use the `lapply` function to compute characteristics of the cohorts. For example if we wanted to find out the number of students from each cohort we could run the following code:

```
lapply(cohorts, length)
```

```
## $Monday
## [1] 5
##
## $Tuesday
## [1] 8
##
## $Wednesday
## [1] 6
```

This code goes through all elements in the object `cohorts` and applies the function `length` to each of them. See `?lapply` and ARP, Section 4.4.1 for more details.

a. Change the names of the list to "Cohort1", "Cohort2" and "Cohort3". (`lapply` is not needed here.). The resulting list is given below:

```
names(cohorts) <- c("Cohort1", "Cohort2", "Cohort3")
cohorts
```

```
## $Cohort1
## [1] 1.0 1.0 5.0 3.3 2.0
##
## $Cohort2
```

```
## [1] 2.0 2.0 4.0 5.0 1.0 1.3 1.7 2.0
##
## $Cohort3
## [1] 3.3 4.0 5.0 1.0 5.0 5.0
```

b. Create a new list where the grades within each cohort are ordered from best to worst. *Hint: ?sort.*

```
cohorts_sort <- lapply(cohorts, sort)
cohorts_sort
```

```
## $Cohort1
## [1] 1.0 1.0 2.0 3.3 5.0
##
## $Cohort2
## [1] 1.0 1.3 1.7 2.0 2.0 2.0 4.0 5.0
##
## $Cohort3
## [1] 1.0 3.3 4.0 5.0 5.0 5.0
```

c. Use the `lapply` function to create a logical vector that indicates whether a student passed the exam or not. A student has passed the exam, if the grade is better than 5. *Hint: Functions that are symbols (e.g. logical operators) only can be handed to* ***`lapply`*** *by wrapping them in quotation marks like this:* ***`"FUN"`****, where* ***`FUN`*** *represents the symbol.* The resulting output is given below:

```
lapply(cohorts, "<", 5)
```

```
## $Cohort1
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
##
## $Cohort2
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
##
## $Cohort3
## [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

d. Use the `lapply` function once again with your result from b. (or c.) as an input to compute the number of students who did not pass in each cohort in a single line of code. *Hint: Think about the properties of logical vectors and how they can be used for calculations..* The resulting output is given below:

```
lapply(lapply(cohorts, ">=", 5), sum)
```

```
## $Cohort1
## [1] 1
##
## $Cohort2
## [1] 1
##
## $Cohort3
## [1] 3
```

Session Info

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.0
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.3.1    fastmap_1.1.1     cli_3.6.1         tools_4.3.1
##  [5] htmltools_0.5.6.1 yaml_2.3.7        rmarkdown_2.25    knitr_1.45
##  [9] xfun_0.40         digest_0.6.33     rlang_1.1.1       evaluate_0.22
```