R Scripting

Exercises for unit 1 - R basics and data structures (I)

Marcus Wurzer

Please solve the following problems!

1. Try to answer quiz questions 1. to 3. at the beginning of the "Data structures" chapter of the coursebook: http://adv-r.had.co.nz/Data-structures.html
2. Coursebook exercises:
   a. Why is `1 == "1"` true? Why is `-1 < FALSE` true? Why is `"one" < 2` false?

      These operators are all functions which coerce their arguments (in these cases) to character, double and character. To enlighten the latter case: "one" comes after "2" in ASCII.

   b. What happens to a factor when you modify its levels?

      ```
      f1 <- factor(letters)
      levels(f1) <- rev(levels(f1))
      ```

      Both, the entries of the factor and also its levels are being reversed:

      ```
      f1
      ```

      ```
      ##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
      ## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
      ```

   c. What does this code do? How do `f2` and `f3` differ from `f1`?

      ```
      f2 <- rev(factor(letters))
      f3 <- factor(letters, levels = rev(letters))
      ```

      ```
      f2 <- rev(factor(letters)) # changes only the entries of the factor
      f3 <- factor(letters, levels = rev(letters)) # changes only the levels of
                                                   # the factor
      ```

      Unlike `f1`, `f2` and `f3` change only one thing. They change the order of the factor or its levels, but not both at the same time.
3. Create the following vectors using `rep()` and `seq()` (as needed):

   ```
   0 -0.1 -0.2 -0.3 -0.4 -0.5 -0.6 -0.7 -0.8 -0.9 -1 -1.1 -1.2 -1.3 -1.4 -1.5
   ```

   ```
   1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
   ```

   ```
   0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
   ```

   ```
   1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9
   ```

   ```
   seq(0, -1.5, -0.1)
   ```

   ```
   ##  [1]  0.0 -0.1 -0.2 -0.3 -0.4 -0.5 -0.6 -0.7 -0.8 -0.9 -1.0 -1.1 -1.2 -1.3 -1.4
   ## [16] -1.5
   ```

```
rep(1:5, 5)
```

```
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

```
rep(0:4, each = 5)
```

```
##  [1] 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
```

```
rep(1:5, 5) + rep(0:4, each = 5)
```

```
##  [1] 1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9
```

4. Create a vector called `colors.6` consisting of the six elements `"red"`, `"brown"`, `"blue"`, `"green"`, `"orange"` and `"black"` (in this order), then use `colors.6`, `rep()` and `seq()` as needed to create the following vector:

```
"red" "brown" "blue" "brown" "blue" "green" "blue" "green" "orange" "green" "orange"
"black"
```

```
colors.6 <- c("red", "brown", "blue", "green", "orange", "black")
colors.6[rep(1:3, 4) + rep(0:3, each = 3)]
```

```
##  [1] "red"    "brown"  "blue"   "brown"  "blue"   "green"  "blue"   "green"
##  [9] "orange" "green"  "orange" "black"
```

5. Construct a vector `x` that contains integers, real numbers, chains of characters, and several `NA` missing values. Test for the positions of the missing values using the `is.na()` function. Produce the subvector where all missing values have been eliminated.

```
(x <- c(1:5, NA, seq(0, 1, 0.2), NA, letters, NA))
```

```
##  [1] "1"   "2"   "3"   "4"   "5"   NA    "0"   "0.2" "0.4" "0.6" "0.8" "1"
## [13] NA    "a"   "b"   "c"   "d"   "e"   "f"   "g"   "h"   "i"   "j"   "k"
## [25] "l"   "m"   "n"   "o"   "p"   "q"   "r"   "s"   "t"   "u"   "v"   "w"
## [37] "x"   "y"   "z"   NA
```

```
idx <- which(is.na(x))
idx
```

```
## [1]  6 13 40
```

```
x[-idx]
```

```
##  [1] "1"   "2"   "3"   "4"   "5"   "0"   "0.2" "0.4" "0.6" "0.8" "1"   "a"
## [13] "b"   "c"   "d"   "e"   "f"   "g"   "h"   "i"   "j"   "k"   "l"   "m"
## [25] "n"   "o"   "p"   "q"   "r"   "s"   "t"   "u"   "v"   "w"   "x"   "y"
## [37] "z"
```

6. Use R to produce a vector containing all integers from 1 to 100 that are not divisible by 7.

```
x <- 1:100
x[x %% 7 != 0]
```

```
##  [1]   1   2   3   4   5   6   8   9  10  11  12  13  15  16  17  18  19  20  22
## [20]  23  24  25  26  27  29  30  31  32  33  34  36  37  38  39  40  41  43  44
## [39]  45  46  47  48  50  51  52  53  54  55  57  58  59  60  61  62  64  65  66
## [58]  67  68  69  71  72  73  74  75  76  78  79  80  81  82  83  85  86  87  88
## [77]  89  90  92  93  94  95  96  97  99 100
```

7. Suppose that `queue <- c("Steve", "John", "Alison", "Liam")` and that `queue` represents a supermarket queue with Steve first in line. Using R expressions, update the supermarket queue successively:

a. Barry arrives.
b. Steve is served and leaves.
c. Pam arrives and talks her way to the front with one item.
d. Barry gets impatient and leaves.
e. Alison gets impatient an leaves. (Do not assume that you know where in the queue Alison is standing.)
f. Use `which()` to find out the position of John in the queue.

```
# a.
queue <- c("Steve", "John", "Alison", "Liam")
(queue <- c(queue, "Barry"))
```

```
## [1] "Steve"  "John"   "Alison" "Liam"   "Barry"
```

```
# b.
(queue <- queue[-1])
```

```
## [1] "John"   "Alison" "Liam"   "Barry"
```

```
# c.
(queue <- c("Pam", queue))
```

```
## [1] "Pam"    "John"   "Alison" "Liam"   "Barry"
```

```
# d.
(queue <- queue[-length(queue)])
```

```
## [1] "Pam"    "John"   "Alison" "Liam"
```

```
# e.
(queue <- setdiff(queue, "Alison"))
```

```
## [1] "Pam"  "John" "Liam"
```

```
# f.
which(queue == "John")
```

```
## [1] 2
```

8. Which of the following assignments will be successful? What will the vectors `x`, `y`, and `z` look like at each stage?

```
rm(list = ls()) # remove (almost) everything in the working environment
x <- 1
x[3] <- 3
y <- c()
y[2] <- 2
y[3] <- y[1]
y[2] <- y[4]
z[1] <- 0
```

```
(x <- 1)
```

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

```
(x[3] <- 3)
```

```
## [1] 3
```

```
(y <- c())
```

```
## NULL
```

```
(y[2] <- 2)
```

```
## [1] 2
```

```
(y[3] <- y[1])
```

```
## [1] NA
```

```
(y[2] <- y[4])
```

```
## [1] NA
```

```
(z[1] <- 0)
```

```
## Error: object 'z' not found
```

9. Show that, when `a` is a scalar (i.e., a vector with just one element) and `x` is a vector, `match(a, x)` is equivalent to `min(which(x == a))`. Compare this with the use of `%in%`.

```
a <- 1
x <- rep(1, 3)
match(a, x) # ?match: match returns a vector of the positions of (first)
```

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

```
# matches of its first argument in its second.
min(which(x == a)) # 3 matches, but only the first one is returned
```

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

```
# because of min()
a %in% x
```

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

```
which(a %in% x)
```

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

10. Use `sample()` to simulate percentage grades of 50 students (0%-100%, only integer values possible) and save the results. Create a second vector containing the gender of these 50 students (again using `sample()`.

    a. Five students' values should be missing because these students were ill at the date of the final exam. Randomly choose five of the 50 percentage grades and replace them by NA's.
    b. Compute the overall mean percentage and the groupwise percentages for males and females, respectively.
    c. Convert the percentage grades into a factor `grades` that contains the student's grades according to the Austrian grading system (from "Sehr gut" to "Nicht genügend"). Use the standard grading system of FH Technikum:
        - $< 50\%:$ *Nicht genügend*
        - $>= 50\% - < 63\%:$ *Genügend*
        - $>= 63\% - < 75\%:$ *Befriedigend*
        - $>= 75\% - < 88\%:$ *Gut*
        - $>= 88\%:$ *Sehr gut*
    d. Compute the grade average for the whole class.

```
x <- sample(0:100, 50, replace = TRUE)
y <- sample(c("m", "f"), 50, replace = TRUE)

# a.
```

```r
x[sample(1:50, 5)] <- NA
x
```

```
##  [1]  87  NA  11  93  NA  72  66  26  69  47  15  69  21   9  73  47  29  68  13
## [20]  57  62  37  27  56  38  55  43  NA  74  94  NA  43  54  55  53  10  82  34
## [39]   4 100   6   9  93  27  22  37  43   0  16  NA
```

```r
# b.
mean(x, na.rm = TRUE) # overall mean
```

```
## [1] 45.46667
```

```r
mean(x[y == "f"], na.rm = TRUE) # mean of females
```

```
## [1] 41.29412
```

```r
mean(x[y == "m"], na.rm = TRUE) # mean of males
```

```
## [1] 48
```

```r
# c.
# Note: We have to set the argument right = TRUE to get intervals closed on
# the right in order to get the correct solution. Otherwise the percentage
# grades that are equal to a breakpoint would result in getting
# a wrong grade (e.g., "Genügend" for 63% instead of "Befriedigend).
# Because of that, we also have to set the argument include.lowest = TRUE,
# otherwise there would be not matching grade for exactly 0 points.
grades <- cut(x, breaks = c(0, 50, 63, 75, 88, 100), include.lowest = TRUE,
              right = TRUE, labels = c("Nicht genügend", "Genügend",
                                       "Befriedigend", "Gut", "Sehr gut"))
grades
```

```
##  [1] Gut            <NA>           Nicht genügend Sehr gut       <NA>
##  [6] Befriedigend   Befriedigend   Nicht genügend Befriedigend   Nicht genügend
## [11] Nicht genügend Befriedigend   Nicht genügend Nicht genügend Befriedigend
## [16] Nicht genügend Nicht genügend Befriedigend   Nicht genügend Genügend
## [21] Genügend       Nicht genügend Nicht genügend Genügend       Nicht genügend
## [26] Genügend       Nicht genügend <NA>           Befriedigend   Sehr gut
## [31] <NA>           Nicht genügend Genügend       Genügend       Genügend
## [36] Nicht genügend Gut            Nicht genügend Nicht genügend Sehr gut
## [41] Nicht genügend Nicht genügend Sehr gut       Nicht genügend Nicht genügend
## [46] Nicht genügend Nicht genügend Nicht genügend Nicht genügend <NA>
## Levels: Nicht genügend Genügend Befriedigend Gut Sehr gut
```

```r
# d.
(grades.num <- as.numeric(grades)) # wrong order! Sehr gut == 5, Gut == 4 etc.
```

```
##  [1]  4 NA  1  5 NA  3  3  1  3  1  1  3  1  1  3  1  1  3  1  2  2  1  1  2  1
## [26]  2  1 NA  3  5 NA  1  2  2  2  1  4  1  1  5  1  1  5  1  1  1  1  1  1 NA
```

```r
(grades.num <- 6 - grades.num) # reverses order to get the correct numeric
```

```
##  [1]  2 NA  5  1 NA  3  3  5  3  5  5  3  5  5  3  5  5  3  5  4  4  5  5  4  5
## [26]  4  5 NA  3  1 NA  5  4  4  4  5  2  5  5  1  5  5  1  5  5  5  5  5  5 NA
```

```r
# representation
mean(grades.num, na.rm = TRUE)
```

```
## [1] 4.044444
```

11. Why do we get a missing value after this sequence of operations?

```
grades <- c("Sehr Gut", "Gut", "Befriedigend", "Gut") # e.g., grades of four pupils
grades <- factor(grades, levels = c("Sehr gut", "Gut", "Befriedigend",
                                    "Genügend", "Nicht genügend"))
grades
```

```
## [1] <NA>         Gut          Befriedigend Gut
## Levels: Sehr gut Gut Befriedigend Genügend Nicht genügend
```

Because R differentiates beween upper- and lowercase letters. "Sehr **g**ut" != "Sehr **G**ut"