Exercises for Recap Session 1
Possible solutions

2024-04-11

Exercise 1: Basic object types I

1. Create a vector containing the numbers 2, 5, 2.4 and 11.

```r
ex1_vec <- c(2, 5, 2.4, 11)
```

2. Replace the second element with 5.9.

```r
ex1_vec[2] <- 5.9
ex1_vec
```

```
[1]  2.0  5.9  2.4 11.0
```

3. Add the elements 3 and 1 to the beginning, and the elements "8.0" and "9.2" to the end of the vector.

```r
va_1 <- c(3, 1)
va_2 <- c("8.0", "9.2")
ex1_vec_extended <- c(va_1, ex1_vec, va_2)
ex1_vec_extended
```

```
[1] "3"   "1"   "2"   "5.9" "2.4" "11"  "8.0" "9.2"
```

4. Create a vector with the numbers from -8 to 9 (step size: 0.5)

```r
ex1_vec_4 <- seq(-8, 9, by = 0.5)
ex1_vec_4
```

```
 [1] -8.0 -7.5 -7.0 -6.5 -6.0 -5.5 -5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -
1.0
[16] -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5
[31]  7.0  7.5  8.0  8.5  9.0
```

5. Compute the square root of each element of the first vector using vectorisation.

```r
sqrt(ex1_vec_4)
```

```
Warning in sqrt(ex1_vec_4): NaNs produced
```

```
 [1]       NaN       NaN       NaN       NaN       NaN       NaN       NaN
 [8]       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[15]       NaN       NaN 0.0000000 0.7071068 1.0000000 1.2247449 1.4142136
[22] 1.5811388 1.7320508 1.8708287 2.0000000 2.1213203 2.2360680 2.3452079
[29] 2.4494897 2.5495098 2.6457513 2.7386128 2.8284271 2.9154759 3.0000000
```

6. Create a character vector containing then strings `"Number_1"` to `"Number_5"`. Use suitable helper functions to create this vector quickly.

```r
ex1_char_vec <- paste0("Number_", seq(1, 5))
ex1_char_vec
```

```
[1] "Number_1" "Number_2" "Number_3" "Number_4" "Number_5"
```

Exercise 2: Basic object types II

Consider the following vector:

```r
ex_2_vec <- c(1.9, "2", FALSE)
```

1. What is the type of this vector? Why?

```r
typeof(ex_2_vec)
```

```
[1] "character"
```

Atomic vectors only contain objects of the same type, and there is a hierarchy. Elements that themselves are of a type lower in the hierarchy are coerced to the same type as the object highest in the hierarchy. The hierarchy is as as follows:

1. `character`
2. `double`
3. `integer`
4. `logical`

Therefore, the type of `ex_2_vec` is `character`. The underlying reason is that you can, for instance, always transform a `double` value into a `character` but not vice versa.

    2. What happens if you coerce this vector into type integer? Why?

```
as.integer(ex_2_vec)
```

```
Warning: NAs introduced by coercion
```

```
[1]  1  2 NA
```

Because `integer` is lower in the hierarchy than `character`, the transformation is not straight-forward. By coincidence, the first two elements can actually be coerced into integers (albeit maybe not with the expected result), but there is no way you can transform the logical value `FALSE` into an integer, which is why a missing value is produced.

    3. What does `sum(is.na(x))` tell you about a vector `x`? What is happening here?

```
x <- c(1,2,3,NA,NA,8)
```

First, `is.na(x)` creates a vector with logical values indicating whether a value of the original vector is missing (i.e. `NA`):

```
is.na(x)
```

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

Then, `sum()` computes the sum over this vecor of boolean values:

```
sum(is.na(x))
```

```
[1] 2
```

Here, `TRUE` counts as one and `FALSE` as zero, so `sum()` gives the number of cases in which `is.na(x)` has evaluated to `TRUE`:

4. Is it a good idea to use `as.integer()` on double characters to round them to the next integer? Why (not)? What other ways are there to do the rounding?

No, because `as.integer()` is not acutally rounding numbers (as, for example, `as.integer(2.1)` would make you think), but only removing the decimal part of the number:

```r
as.integer(2.9) # you might expect 2...
```

```
[1] 2
```

Better use `round()`:

```r
round(2.9)
```

```
[1] 3
```

Exercise 3: Define a function

Create functions that take a vector as input and returns:

1. The last value.

```r
get_last_val <- function(x){
  last_val <- x[length(x)]
  return(last_val)
}
```

2. Every element except the last value and any missing values.

```r
get_beginning <- function(x){
  beginning <- x[-length(x)] # Removes last value
  na_positions <- which(is.na(beginning)) # Get positions of NA values
  beginning_nonas <- beginning[-na_positions] # Removes these values
  return(beginning_nonas)
}
```

3. Only even numbers.

   Hint: Use the operation `x %% y` to get the remainder from diving `x` by `y`, the so called 'modulo y'. For even numbers, the modulo 2 is zero.

```r
get_even <- function(x){
  modulo_2s <- x%%2 # Module 2 is zero for even numbers only
  even_nbs <- x[modulo_2s==0] # Keep only those for which modulo 2 is zero
  na_positions <- which(is.na(even_nbs)) # Get positions of NA values
  even_nbs_nonas <- even_nbs[-na_positions] # Removes these values
  return(even_nbs_nonas)
}
```

Apply your function to the following example vector:

```r
ex_3_vec <- c(1, -8, 99, 3, NA, 4, -0.5, 50)
```

```r
get_last_val(ex_3_vec)
```

```
[1] 50
```

```r
get_beginning(ex_3_vec)
```

```
[1]   1.0 -8.0 99.0  3.0  4.0 -0.5
```

```r
get_even(ex_3_vec)
```

```
[1] -8  4 50
```

Exercise 4: Lists

1. Create a list that contains three elements called `'a'`, `'b'` and `'c'`. The first element should correspond to a double vector with the elements `1.5`, `-2.9` and `99`. The second element should correspond to a character vector with the elments `'Hello'`, `'3'`, and `'EUF'`. The third element should contain three times the entry `FALSE`.

```r
ex_4_list <- list(
  'a' = c(1.5, -2.9, 99),
  'b' = c('Hello', "'3'", 'EUF'),
  'c' = rep(FALSE, 3)
)
```

2. Transform this list into a `data.frame` and a `tibble`. Then apply `str()` to get information about the respective structure. How do the results differ?

```
ex_4_df <- as.data.frame(ex_4_list)
ex_4_tb <- tibble::as_tibble(ex_4_list)
str(ex_4_list)
```

```
List of 3
 $ a: num [1:3] 1.5 -2.9 99
 $ b: chr [1:3] "Hello" "'3'" "EUF"
 $ c: logi [1:3] FALSE FALSE FALSE
```

```
str(ex_4_df)
```

```
'data.frame':   3 obs. of  3 variables:
 $ a: num  1.5 -2.9 99
 $ b: chr  "Hello" "'3'" "EUF"
 $ c: logi  FALSE FALSE FALSE
```

```
str(ex_4_tb)
```

```
tibble [3 x 3] (S3: tbl_df/tbl/data.frame)
 $ a: num [1:3] 1.5 -2.9 99
 $ b: chr [1:3] "Hello" "'3'" "EUF"
 $ c: logi [1:3] FALSE FALSE FALSE
```

`str()` only differs with regard to the first line describing the type.

Exercise 5: Data frames and the study semester distribution at EUF

The package `DataScienceExercises` contains a data set called `EUFstudentsemesters`, which contains information about the distribution of study semesters of enrolled students at the EUF in 2021. You can shortcut the data set as follows:

```
euf_semesters <- DataScienceExercises::EUFstudentsemesters
```

1. What happens if you extract the column with study semesters as a vector and transform it into a `double`?

```r
unique(euf_semesters[["Semester"]])
```

```
[1] "6"          "4"          "2"          "8"          "9 or higher"
[6] "7"          "5"          "3"          "1"
```

```r
semesters <- as.double(euf_semesters[["Semester"]])
```

```
Warning: NAs introduced by coercion
```

```r
unique(semesters)
```

```
[1]  6  4  2  8 NA  7  5  3  1
```

We see that the previous entry **"9 or higher"** has been transformed into NA.

2. What is the average study semester of those students being in their 8th or earlier semester?

```r
mean(semesters, na.rm = TRUE)
```

```
[1] 4.177026
```

3. How many students are in their 9th or higher study semester?

```r
sum(euf_semesters$Semester=="9 or higher")
```

```
[1] 469
```

4. What does `typeof(euf_semesters)` return and why?

```r
typeof(euf_semesters)
```

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

It returns `list`, because while `euf_semesters` is a `tibble`, `typeof()` always gives the underlying basic object type. For `tibbles`, this is `list`.