Problem Set 4a

2023-12-09

Part 1: R questions

Question 1: Functions in R

1.

```
count_na <- function(x) {
  mean(is.na(x))
}

mean <- function(x) {
  x / sum(x, na.rm = TRUE)
}

sd <- function(x) {
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
}
```

2.

```
both_na <- function(x,y) {
  which(is.na(x) & is.na(y))
}

both_na(c(NA, 2, NA, 2, NA), c(NA, 2, NA, 4, 5))
```

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

3.

The first line checks whether the give path is that of a directory The second line checks if the path to a given file is readable

4.

The first function checks if the prefix variable and prefix of the string are the same. I would rename this function to “check_prefix” The second function returns null if the length of x is less than or equal to 1. Otherwise, it will return x without the last element. I would rename this “rm_last_letter”. The third function takes the second vector and prints its values in a loop until however long the x variable is. So I would just call this function as “repeater”.

5.

```
fizzbuzz <- function(x) {
  three <- x %% 3 == 0
  five <- x %% 5 == 0

  if (three && five) {
    return("fizzbuzz")
  } else if (three) {
    return("fizz")
  } else if (five) {
    return("buzz")
  } else {
    return(x)
  }
}

fizzbuzz(5)
```

```
## [1] "buzz"
```

6.

For the sign <=

```
temp <- c(5, 23)
cut(temp, breaks = seq(-10,40,10), labels = c("freezing", "cold", "cool", "warm", "ho
t"))
```

```
## [1] cold warm
## Levels: freezing cold cool warm hot
```

For the sign <

```
cut(temp, breaks = seq(-10,40,10), right = FALSE, labels = c("freezing", "cold", "coo
l", "warm", "hot"))
```

```
## [1] cold warm
## Levels: freezing cold cool warm hot
```

The chief advantage would be deciding what the closing values are

7.

Basically it goes to the given letter and checks the value. If there’s no value, then it goes to the next letter

```
x <- "e"
switch_out <- switch(x,
a = ,
b = "ab",
c = ,
d = "cd"
)
switch_out
```

```
## NULL
```

With e, because there's no letter after d in switch, it prints NULL

Question 2: Vectors and lists in R

1.

id.vector() checks if the given vector contains names. Anything other than names, it will return error or false
is.atomic() returns true for values which are also NULL.

2.

```
last <- function(x) x[[length(x)]]
last(1:10)
```

```
## [1] 10
```

```
even_position <- function(x) x[((1:length(x)) %% 2 == 0)]
even_position(2:10)
```

```
## [1] 3 5 7 9
```

```
except_last <- function(x) x[-length(x)]
except_last(1:10)
```

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

```
even <- function(x) x[x %% 2 == 0]
even(2:10)
```

```
## [1]  2  4  6  8 10
```

3.

which is only meant for indices

4.

You will only get an NA in both the scenarios

5.

```
list(1, 2, list(3, 4), list(5, 6))
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [[3]][[1]]
## [1] 3
##
## [[3]][[2]]
## [1] 4
##
##
## [[4]]
## [[4]][[1]]
## [1] 5
##
## [[4]][[2]]
## [1] 6
```

Basically here we can see the diagram but index wise. For example, in 3rd list, we expect to see 3 in 1st position and 4 in 2nd position. And that is the output. [[3]][[1]] = 3 & [[3]][[2]] = 4.

Similarly for this:

```
list(list(list(list(list(list(1))))))
```

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

Question 3: Iteration in R

1.

I am commenting the code here because RMarkdown is having difficulty executing it during export. The code is working perfectly fine when running it directly. # 1.

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflic
ts to become errors
```

```
#mean_col <- vector("double", length(mtcars))
#for (i in seq_along(mtcars)){
#  mean_col[[i]] <- mean(mtcars[[i]])
#}

#mean_col
```

2.

```
library(nycflights13)

flight_col_type <- vector("character", length(flights))
for (i in seq_along(flights)){
  flight_col_type[[i]] <- typeof(flights[[i]])
}

flight_col_type
```

```
##  [1] "integer"   "integer"   "integer"   "integer"   "integer"   "double"
##  [7] "integer"   "integer"   "double"    "character" "integer"   "character"
## [13] "character" "character" "double"    "double"    "double"    "double"
## [19] "double"
```

3.

```
data("iris")

unique_vals <- vector("integer", length(iris))
for (i in seq_along(iris)){
  unique_vals[[i]] <- unique(iris[[i]]) %>% length()
}

unique_vals
```

```
## [1] 35 23 43 22  3
```

4.

```
normals <- vector("list", 4)
input_means <- c(-10, 0, 10, 100)
for (i in seq_along(normals)){
  normals[[i]] <- rnorm(10, mean = input_means[[i]])
}

normals
```

```
## [[1]]
##  [1]  -8.466508  -8.068790 -11.124191  -9.784619  -8.849273  -8.157861
##  [7]  -8.370828 -11.056264 -10.690167 -10.635772
##
## [[2]]
##  [1]  1.1491989 -1.3063385  1.1201847  0.5953820 -0.9590045 -0.1163036
##  [7] -1.0872474 -0.4681493 -0.1530616 -1.1911235
##
## [[3]]
##  [1] 10.258453  9.749757  8.406502  9.425893  9.327208 10.425241 10.676786
##  [8]  8.365681 10.056112  9.880508
##
## [[4]]
##  [1] 100.88161  99.59545  99.16019 100.41813  99.94080 102.01057 101.42162
##  [8] 101.36020 100.62568 100.77163
```

2.

```
out <- ""
for (x in letters) {
out <- stringr::str_c(out, x)
}

str_c(letters, collapse = "")
```

```
## [1] "abcdefghijklmnopqrstuvwxyz"
```

```
x <- sample(100)
sd <- 0
for (i in seq_along(x)) {
  sd <- sd + (x[i] - mean(x)) ^ 2
}
sd <- sqrt(sd / (length(x) - 1))
sd
```

```
##   [1] 58.44656 58.44552 58.44829 58.45520 58.45399 58.44984 58.45762 58.44690
##   [9] 58.45278 58.45589 58.45710 58.44570 58.45813 58.44379 58.45157 58.44915
##  [17] 58.45382 58.44881 58.45122 58.45261 58.46021 58.44725 58.45675 58.45796
##  [25] 58.45036 58.44742 58.44932 58.45433 58.45053 58.44760 58.45330 58.45295
##  [33] 58.45727 58.44898 58.45572 58.45105 58.44500 58.44535 58.45312 58.46055
##  [41] 58.44950 58.44362 58.46073 58.45883 58.45468 58.44811 58.45174 58.45606
##  [49] 58.45537 58.45364 58.45192 58.44466 58.45693 58.44777 58.45831 58.44414
##  [57] 58.45243 58.45969 58.44397 58.45986 58.44449 58.46004 58.45347 58.45451
##  [65] 58.45779 58.44708 58.46038 58.45848 58.44621 58.45071 58.44863 58.45744
##  [73] 58.45485 58.44673 58.44518 58.45502 58.45641 58.45001 58.44639 58.44794
##  [81] 58.44967 58.45623 58.45658 58.45226 58.45140 58.45865 58.44483 58.44431
##  [89] 58.44587 58.45554 58.45952 58.45900 58.45416 58.45019 58.45917 58.45934
##  [97] 58.44604 58.44846 58.45088 58.45209
```

```
sd(x)
```

```
## [1] 29.01149
```

```
x <- runif(100)
out <- vector("numeric", length(x))
out[1] <- x[1]
for (i in 2:length(x)) {
out[i] <- out[i - 1] + x[i]
}
#out

cumsum(x)
```

```
##   [1]  0.002751895  0.386732026  1.061036330  1.538528400  1.569591426
##   [6]  1.696082623  2.414270726  2.915805437  3.781416251  4.044766005
##  [11]  4.335504861  4.704562576  5.589072623  5.818243321  6.300543053
##  [16]  7.262113679  7.534545805  7.870119262  8.726222310  8.768429090
##  [21]  9.468428268 10.120278180 10.654671059 11.450677463 11.791970783
##  [26] 12.128867655 12.244675608 12.685900005 13.424320504 13.992375055
##  [31] 14.047831808 14.160701048 14.242598085 14.325165943 14.869472727
##  [36] 15.315223568 16.313114183 16.519170973 16.759492670 16.926259242
##  [41] 17.145390535 17.680570508 18.606128491 18.890321264 19.666078092
##  [46] 20.487351892 20.694462185 20.784309164 20.883492813 21.682584886
##  [51] 22.404634390 23.002552533 23.541495207 24.046548508 24.808697224
##  [56] 25.654187471 25.895182210 26.007742845 26.896271522 26.975369875
##  [61] 27.703064468 28.605605339 29.032714652 29.380161992 30.336514288
##  [66] 30.573600639 30.790082307 31.687731918 32.150379423 32.384242145
##  [71] 32.572613710 32.582576826 32.802506738 33.542476448 34.314972772
##  [76] 35.182262318 35.746522639 36.415290885 37.019304047 37.087032185
##  [81] 38.006285882 38.905731661 39.465357352 39.605662641 39.942325837
##  [86] 40.600198998 41.340857184 41.776007163 42.573231734 42.890857161
##  [91] 43.200396086 43.213517818 43.315622288 43.946342970 44.015821781
##  [96] 44.884465055 45.722325163 46.348859564 46.813385226 47.353963256
```

3.

```
output <- vector("integer", 0)
for (i in seq_along(x)) {
output <- c(output, lengths(x[[i]]))
}
output
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

In general, with programming we know that preallocating the output space is a good practice and so this has to mean that preallocating will have better efficiency rather than increasing the length of a vector at each step. After googling, I could find that generally you performance goes up by around 30% by preallocating which is pretty significant especially if you want to deal with large datasets.

Part 2: Your Project

I have already worked on including graphs, sections, and other such information in R markdown before and it was pretty easy. Although I would still prefer a word document because it’s easier to format things like font size, image size and exporting the document(for some reason RMarkdown has difficulty running code during export process even though it has no issues running it directly). I am yet to finish up with the content of the report. If I feel like I can get away with a good looking report on R markdown, the perfect. Otherwise, I will type it up on a word document.