Problem Sheet 2 - CT5102

Lists, Functions and Functionals

The goal of these short exercise is to practice key ideas from the lecture.

In most cases, the answers are provided as part of the output, the challenge is to write R code that will generate the results.

1. In R, a data frame (which will be introduced in later chapters) can be converted to a list, using the function **`as.list()`**. There is a useful data frame in R known as **`mtcars`**, and we fill first convert this to a list, and explore it's structure.

```
cars <- as.list(mtcars)
str(cars)
```

```
## List of 11
##  $ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num [1:32] 160 160 108 258 360 ...
##  $ hp  : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
##  $ vs  : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
```

Notice that this is a list of eleven elements, where each element is a feature of a specific car. For example **`mpg`** represents the miles per gallon for each car, and **`disp`** stores the engine

size, or displacement. The list contents are eleven numeric vectors, each of size 32, and these can be viewed as parallel arrays, where all data in the 1st of each vector refers to car number one, and all data in final location refers to the final car.

The aim of the exercise is to generate a list to contain the mean value for **mpg** and **disp**. Use the following code to create the list structure.

```
result <- list(mean_mpg=vector(mode="numeric", length=1),
               mean_disp=vector(mode="numeric", length=1))
str(result)
```

```
## List of 2
##  $ mean_mpg : num 0
##  $ mean_disp: num 0
```

The following shows the calculated result. Note that the **cars** list should first be subsetted to only include those two elements that are required (i.e. **mpg** and **disp**.)

```
str(result)
```

```
## List of 2
##  $ mean_mpg : num 20.1
##  $ mean_disp: num 231
```

You can confirm your results with the following commands in R.

```
mean(cars$mpg)
```

```
## [1] 20.09062
```

```
mean(cars$disp)
```

```
## [1] 230.7219
```

2. Filter the list **sw_people** (87 elements), contained in the library **repurrrsive** to include only those whose height is not unknown. Use an atomic vector **has_height** to filter the list, and populate this vector using a loop structure. This new list (**sw_people1**) should have 81 elements.

```
sum(has_height)
```

```
## [1] 81
```

```
length(sw_people1)
```

```
## [1] 81
```

3. Using a **for** loop over the filtered list **sw_people1** from exercise 2, create a list of people whose height is greater than or equal to 225 inches. The resulting vector should grow as matches are found, as we do not know in advance how many people will be contained in the result. Use the command **characters <- c()** to create the initial empty result vector. The **if** expression may be useful here too, and make sure that the height value is converted to numeric before evaluating.

The following result should be obtained.

```
# These are the characters whos height is >= 225
characters
```

```
## [1] "Chewbacca"   "Yarael Poof" "Lama Su"     "Tarfful"
```

4. Using a **for** loop to iterate over the list **sw_planets** and display those planets (in a character vector) whose diameter is greater than or equal to the mean. Use a pre-processing step that will add a new list element to each planet, called **diameter_numeric**. This pre-preprocessing step can also be used to calculate the mean diameter. Also, use the pre-processing step to keep track of those planets whose diameter is "unknown", and use this information to create an updated list **sw_planets1** that excludes all the values.

You can check your solution against the following values.

```
# The list elements that will be excluded (diameter unknown)
exclude
```

```
##  [1] 37 39 42 44 45 46 47 49 50 51 52 53 54 55 57 59 61
```

```
# The mean diameter
mean_diameter
```

```
## [1] 8935.852
```

```
# The first three and last three planets returned
gte_mean[c(1:3,(length(gte_mean)-2):(length(gte_mean)))]
```

```
## [1] "Alderaan"   "Yavin IV"   "Bespin"     "Muunilinst" "Kalee"
## [6] "Tatooine"
```

5. Based on the list **sw_species**, and given that each species has a **classification**, create the following tabular summary, again using a loop to iterate through the list. Make use of the **table()** function that was covered in Chapter @ref(ch2) to present the summary.

```
# A tabular summary of the types of species
t_species
```

```
## c_species
##  amphibian artificial  gastropod  insectoid     mammal    mammals    reptile
##          6          1          1          1         16          1          3
##  reptilian   sentient    unknown
##          1          1          6
```

6. Write a function **get_even1()** that returns only the even numbers from a vector. Make use of R's modulus function **%%** as part of the calculation. Try and implement the solution as one line of code. The function should transform the input vector in the following way.

```
set.seed(200)
v <- sample(1:20,10)
v
```

```
##  [1]  6 18 15  8  7 12 19  5 10  2
```

```
v1 <- get_even1(v)
v1
```

```
## [1]  6 18  8 12 10  2
```

7. Write a similar function **get_even2()** that takes a second parameter **na.omit**, with a default of **FALSE**. If **na.omit** is set to **TRUE**, the vector is pre-processed in the function to remove all **NA** values before doing the final calculation.

```
set.seed(200)
v <- sample(1:20,10)
i <- c(1,5,7)
v[i] <- NA
v
```

```
##  [1] NA 18 15  8 NA 12 NA  5 10  2
```

```
v1 <- get_even2(v)
v1
```

```
## [1] NA 18  8 NA 12 NA 10  2
```

```
v2 <- get_even2(v,na.omit=TRUE)
v2
```

```
## [1] 18  8 12 10  2
```

8. Which one of the following three functions calls to **fn_test()** will not work. Why?

```
# The function
fn_test <- function(a, b, c){
  a+b
}
# Call 1
fn_test(1,2)
# Call 2
```

```
fn_test(c=1,2)
# Call 3
fn_test(b=1,10)
```

9. What will be the output from the following function call?

```
a <- 100


env_test <- function(b,c=20){
  a+b+c
}


env_test(1)
```

10. Use **lapply()** followed by an appropiate post-processing function call, to generate the following output, based on the input list.

```
# Create the list that will be processed by lapply
l1 <- list(a=1:5,b=100:200,c=1000:5000)
```

```
# The result is stored in ans
ans
```

```
##    a    b    c
##    3  150 3000
```