Scientific Programming – Intorduction to R, CRAN and Bioconductor

Exercises:

1) What is the value of `pi` to 3 decimal places?
- See the help for `round` `?round`

2) How can we a create a sequence from 2 to 20 comprised of 5 equally-spaced numbers?
- Check the help page for seq `?seq`

3) Create a *variable* containing 1000 random numbers with a *mean* of 2 and a *standard deviation* of 3
- What is the maximum and minimum of these numbers?
- What is the average?
- Hint: see the help pages for functions `min`, `max` and `mean`
- What is the sum of the 1000 random numbers?
- What are the natural logarithms of these 1000 random numbers? (Hoping that none of them is zero)

4) Try to let R compute the natural logarithm of zero and see what result you get …

5) Try to let R compute 1 divided by 0 and see what result you get …

6) What do you get if you try to convert the string “Hello World” to a numeric value?

7) The function `seq` has some related functions, see “`?seq`”. Find out what `seq_along` does and come up with an example.
- What will you obtain if you execute “`seq_along(rnorm(12))`”?
- How can you achieve the same result using `seq` and `length`?

8) Pay attention to the fact that all elements of a vector must be of the same type. This may at times appear to lead to strange effects (which can however be logically explained). Try the following:
Define the following vector in R:

```
x <- c(TRUE, FALSE, FALSE, NaN, TRUE)
```

What to you obtain when you want to display the content of the vector by simply typing “`x`”?
Can you imagine why this is the case?

9) You can also read a file from the Web via HTTP or HTTPS. Read the tab-separated table as a data.frame from the following source:
https://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt
- The table describes what sort of single nucleotide mutations are caused by mutational processes like smoking, UV light and others. We don’t need to know about the biological background for the moment. We will just “play” with the table.
- Look at the file in a web browser (or download it and use a text editor), so that you see how the original looks like.
- Load the file as a data.frame, but:

- Make sure the first line of the file is used as column names, i.e., as header, for the data.frame!
- The table has no row names. But the third column does contain unique keys, so it can be used as rownames. Find out how this can be done with `read.table`.
- If you load the table correctly, it should look like this:

```
> mutsign[1:4,1:4]
        Substitution.Type Trinucleotide Signature.1  Signature.2
A[C>A]A               C>A           ACA 0.011098326 6.827082e-04
A[C>A]C               C>A           ACC 0.009149341 6.191072e-04
A[C>A]G               C>A           ACG 0.001490070 9.927896e-05
A[C>A]T               C>A           ACT 0.006233885 3.238914e-04
```

- What are the dimensions of the data frame?
- What columns are available? Find out about the structure of the data frame ...
- What are the names of the columns?
- There are three different ways of getting the content of a specific column. Figure out which (in case you don't know them already). Try all of them. What result do you get (data type)? Is there any difference in what you get as a result between the three ways?
- Try to get the column content of both, a column containing numbers and a column containing text. What do you get back?
- HINT: see the `class`, `dim`, `ncol`, `nrow` and `colnames` functions. Can you find a function for getting the structure of a data frame?

10) Take the data.frame from the previous task and do the following:

- Extract the numeric data columns (using `ncol` for the total number of columns!) and write them into a new data object.
- You will probably notice a problem: the original file seems to contain additional <tab>-symbols for empty columns at the end of each line, so that you obtained 7 columns more than expected … remove these columns also.
- Make sure this new data object is a numeric matrix!
- If everything works fine, you should obtain the following:

```
> mutsign2[1:4,1:4]
        Signature.1  Signature.2 Signature.3 Signature.4
A[C>A]A 0.011098326 6.827082e-04  0.02217231      0.0365
A[C>A]C 0.009149341 6.191072e-04  0.01787168      0.0309
A[C>A]G 0.001490070 9.927896e-05  0.00213834      0.0183
A[C>A]T 0.006233885 3.238914e-04  0.01626515      0.0243
> dim(mutsign2)
[1] 96 30
> class(mutsign2)
[1] "matrix"
```

- For extracting a specific column from a matrix, only two ways will work; one of the three ways which work for data.frames will NOT work for a matrix. Find out which!
- Extract the first three columns only for rows which have a value of > 0.02 for Signature 1:
  - How many rows does the result have?
  - Would you say that Signatures 2 and 3 are similar to Signature 1? You can easily check the entire output against the threshold of 0.02!

11) Working with factors. Define a factor using a well defined set of levels. I suggest to use a set of levels of numeric values, because it illustrates an important issue about factors, e.g.

```
myfac <- factor(c(1, 2, 3, 1, 2, 1), levels=1:4)
```

- Try conversion to a vector and verify what type of elements the output vector has. Numeric value?
- Try to verify what type of elements the level are composed of. Numeric values?
- Try to set the first element of the factor to a value not present in the levels. What happens? How does the factor look like afterwards?
- Now convert the factor obtained from the previous task into a vector. What do you obtain?

12) This exercise is IMPORTANT, because you will encounter this problem more often for named vectors, matrices or data.frame objects!
Create a vector with several arbitrary but unique elements[1], e.g.,

```
myvec <- 1:10
```

Set the element names for this vector using also numeric values in reverse order, e.g.,

```
names(myvec) <- 10:1
```

or better:

```
names(myvec) <- rev(myvec)
```

(As always, check how the new function works: `?rev`)

- Have a look at the names vector and then verify what element you get with **`myvec[3]`**
- Do you get the same element when you use `myvec["3"]`???
- Can you imagine what happened? Be aware of this behavior when you work with named vector or row/column names of tables where the names are, for example, numeric gene IDs from the NCBI database!!!
- Example: The NCBI/Entrez gene ID of the gene AKT3 is 10000. Imagine a typical gene expression matrix where the row names are gene IDs and the column names are sample IDs/names. Say the gene expression matrix uses NCBI/Entrez gene IDs for the row. If you want extract the gene expression vector (row) for gene AKT3, would you obtain it using

  ```
  expmatrix[10000,]
  ```

  ?
  If yes, justify why.
  If not, what would you need to use instead?

13) Matrix and data.frame

- Create the data frame of patient data from the lecture and convert it into a matrix. What do you obtain? Of what basic data type are the single elements of the new matrix?
- Now take the data fram from the lecture and delete the column with the tumor name. Convert the obtained data.frame again into a matrix. Do you get a matrix with the same basic data type as before?

---

1 This works of course also with non-unique elements, but using unique elements in this exercise will make it easier to observe the undesired effect.

14) Create the list mylist from the lecture (or any other list)

- Try to better understand the difference between

  `mylist[1]`

  and

  `mylist[[1]]`

  Use the command "`class`" to figure out what exactly you get back from these two ways of addressing list elements!

15) Instead of `if` … `else if` … `else if` … `else if` … else where the conditions just check the different values a given variable can have, one can use "`switch`". You can find a short tutorial here:
https://www.tutorialgateway.org/r-switch-statement/

16) Con you predict the outcome of the following code before executing it?

```
xs <- c(1, 2, 3)
for (x in xs) {
  xs <- c(xs, x * 2)
}
xs
```