Introductory Workshop in Bioinformatics - Part 2

Dr. Jenny Russ

5 September 2022

Setting up RStudio / Installing required packages

```
install.packages("tidyverse")
```

```
## Warning: Paket 'tidyverse' wird gerade benutzt und deshab nicht installiert
```

```
install.packages("ggpubr")
```

```
## Warning: Paket 'ggpubr' wird gerade benutzt und deshab nicht installiert
```

Download files required for this tutorial

- Go to: https://github.com/jentiger82/BioinformaticsWorkshop2022
- Download all files and save them in an easy to find folder

Getting to know R

- Run: library(tidyverse)
- For the first exercise load files SEQ1.txt and SEQ2.txt
- You can replace the web address below with your local file path to the files: “D:/Bioinformatics_Seminar_2022/SEQ1.txt”

```
seq1 <- read_file(
  "https://raw.githubusercontent.com/jentiger82/BioinformaticsWorkshop2022/main/SEQ1.txt")
seq2 <- read_file(
  "https://raw.githubusercontent.com/jentiger82/BioinformaticsWorkshop2022/main/SEQ2.txt")
seq_1 <- read_file("D:/Bioinformatics_Seminar_2022/SEQ1.txt")
```

We want to know if sequence 1 is the same as sequence 2

```
#This is a comment.

#Printing seq1 and seq2
seq1
```

```
## [1] "agagacacagccgaaccccaggcagttttggccggtttgccaggttccccgttaccgtcctt"
```

```
seq2
```

```
## [1] "agagacacagccgaaccccaggcagttttggccgctttgccaggttccccgttaccgtcctt"
```

```
#Are they equal?
seq1==seq2
```

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

5 basic data structures in R

- Character
- Numeric
- Integer
- Logical
- Complex

Character

```r
# Holds "string" data. For example:
"Hello World"
```

```
## [1] "Hello World"
```

```r
# You can ask R to tell you what type of data you have:
class("Hello World")
```

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

Numbers

```r
# Numbers can be integers or numerics (i.e., floats) in R. For example:
1
```

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

```r
2.2
```

```
## [1] 2.2
```

```r
3E100
```

```
## [1] 3e+100
```

Numbers

```
# The default for R is to make every number a numeric
class(1)
```

```
## [1] "numeric"
```

```
# Which means that we have hidden decimal points behind the 1 (1.00 for example)
```

But what if we wanted specifically an integer?

```r
1L
```

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

```r
class(1L)
```

```
## [1] "integer"
```

Challenge Question 1

```r
# Is this a numeric or a character?
"2"
```

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

Challenge Question 1

```r
# Is this a numeric or a character?
"2"
```

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

```r
class("2")
```

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

Logicals

```
# Logicals are either True of False. For example
TRUE
```

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

```
FALSE
```

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

```
class(TRUE)
```

```
## [1] "logical"
```

```
class(FALSE)
```

```
## [1] "logical"
```

Challenge Question 2

```
# R is *case sensitive*.
# With that knowledge, what do you think the below would give us?

#class(true)
```

Challenge Question 2

```
# R is *case sensitive*.
# With that knowledge, what do you think the below would give us?

#class(true)
```

Output: Error: object ‘true’ not found

Data Structure Classes cont..

```r
# So far we have only been using class()
# But we can also use a function to see the structure of any R object:
str(1)
```

```
##  num 1
```

```r
str("Hello World!")
```

```
##  chr "Hello World!"
```

```r
str(TRUE)
```

```
##  logi TRUE
```

Challenge Question 3

```
# R is an object oriented language, and it tends to put everything in some sort of class.
# What would the below tell us?
```

str(class)

Challenge Question 3

```
# R is an object oriented language, and it tends to put everything in some sort of class.
# What would the below tell us?
str(class)
```

```
## function (x)
```

Simple Data Structures Methods

Arithmetic Methods

```
# * Addition
1+1
```

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

```
# * Subtraction
1-1
```

```
## [1] 0
```

```
# * Multiplication
2*2
```

```
## [1] 4
```

```
# * Division
10/2
```

```
## [1] 5
```

Equivalence Comparisons

```r
# Equivalence comparisons are a way to check if any two objects are the same
#Does 1 equal 1?
1==1
```

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

```r
#Does "Hello" equal "World?"
"Hello" == "World"
```

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

Equivalence Comparisons

```
#We can also invert an equivalancy comparison to check if two objects are *not* equal
#Does 1 not equal 1?
1!=1
```

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

Mathematical Comparisons

```
#For numeric data types, mathematical comparisons can also be made
#Is 1 less than 100?
1<100
```

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

```
#Is 2+2 greater than 2^2
2+2>2^2
```

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

Challenge Question 4

```r
# Arithmetic in R follows PEMDAS
4+5*3
```

```
## [1] 19
```

```r
# vs
(4+5)*3
```

```
## [1] 27
```

```r
# What would we get with the following?
```

4+5*3>(4+5)*3

Challenge Question 4

```r
# Arithmetic in R follows PEMDAS
4+5*3
```

```
## [1] 19
```

```r
# vs
(4+5)*3
```

```
## [1] 27
```

```r
# What would we get with the following?
4+5*3>(4+5)*3
```

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

Variables

```r
# Variables hold objects that are assigned to them.
a <- 1
# They can be identical to the object assigned to them
a <- 1
b <- a
#does b equal 1?
b==1
```

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

Variables enable complex operations on data

```
h <- 2^100
i <- h/3E100
j <- 1E5
k <- j^(-1*i)
#is k greater than 1?
k > 1
```

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

Complex Data Structures

Vectors

```r
# A Vector is an ordered collection of either numerics, characters, or logicals
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```r
c(TRUE, FALSE, TRUE)
```

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

```r
c("hello", "world")
```

```
## [1] "hello" "world"
```

Challenge Question 5

```
# A vector in R has a limitation: they can only allow one type of class in each vector.
# If you add more than one type of class, it will default to a character.
# With that said, what would the following give you?
```

class(c("puppies", 2))

Challenge Question 5

```r
class(c("puppies", 2))
```

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

Challenge Question 6

```
# I can change the following vector in my variable to all numbers
my_num <- as.numeric(c("2", 2))
class(my_num)
```

```
## [1] "numeric"
```

```
# But what would happen if I tried to do the following?
```

as.numeric(c("salt", "2")) print(as.numeric(c("salt", "2")))

Challenge Question 6

```r
print(as.numeric(c("salt", "2")))
```

```
## Warning in print(as.numeric(c("salt", "2"))): NAs durch Umwandlung erzeugt
```

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

```r
class(as.numeric(c("salt", "2")))
```

```
## Warning: NAs durch Umwandlung erzeugt
```

```
## [1] "numeric"
```

Vectors Continued

```r
# Vectors can also have a vector of names which describe each element
grades <- c(98, 95, 82)
names(grades) <- c("Jimmy", "Alice", "Susan")
grades
```

```
## Jimmy Alice Susan
##    98    95    82
```

```r
# Elements from a vector can be accessed using the index of the desired data. Lets say I wanted to know wh
grades[2]
```

```
## Alice
##    95
```

```r
# But lets say that I didn't know Alice was the second index
grades["Alice"]
```

```
## Alice
##    95
```

Vectors Continued

```
# If you want to add a series of numbers to a vector of integers, you can do the following:
my_ints <- 1:10
my_ints
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Lists

```
# A list is an ordered collection of any objects. This differs from vectors, because vectors can only be o
my_list <- list(1, "b", TRUE, c(1,2,3))
# Lists can also have names
names(my_list)=c("My #", "My Letter", "My logical", "My_num_Set")
my_list
```

```
## $`My #`
## [1] 1
##
## $`My Letter`
## [1] "b"
##
## $`My logical`
## [1] TRUE
##
## $My_num_Set
## [1] 1 2 3
```

Lists continued

```
# The elements within a list can be accessed by using numeric indexes or by the element name
my_list[[2]]
```

```
## [1] "b"
```

```
my_list[["My_num_Set"]]
```

```
## [1] 1 2 3
```

```
my_list$My_num_Set
```

```
## [1] 1 2 3
```

```
#If I did the following, I would get the whole vector within the list
my_list[2]
```

```
## $`My Letter`
## [1] "b"
```

Data Frames

```r
# Data frames are similar to excel sheets. They are 2D arrays which can hold numeric, character, and boole
my_df <- data.frame("students"=c("Jimmy", "Alice", "Susan"),
                    "Grades"=c(98, 95, 82))
my_df
```

```
##   students Grades
## 1    Jimmy     98
## 2    Alice     95
## 3    Susan     82
```

Data Frames

```
#Data frames can be accessed numerically by expressing the row and column of interest.
#What grade did Susan get?
my_df[3,2] #row, column
```

```
## [1] 82
```

```
# They can also be accessed with the $ sign
my_df$Grades
```

```
## [1] 98 95 82
```

```
#What grade did Alice get?
my_df$Grades[2]
```

```
## [1] 95
```

Using Data Frames - Iris data

```
# How does the iris data look like?
#View(iris)
# Get row 1
row1 <- iris[1,]
# Get column 1
col1 <- iris[,1]
# Access a column by name
pl <- iris$Petal.Length

# How to get the value in row 1, column 2?
# How would I get the "species" in row 3?
```

We continue in 15 minutes…

Iris data - solutions

```r
# How to get the value in row 1, column 2?
iris[1,2]
```

```
## [1] 3.5
```

```r
# How would I get the "species" in row 3?
iris$Species[3]
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```