

Introduction to Bioinformatics

Alexandros C. Dimopoulos
alexdem@di.uoa.gr

Master of Science
“Data Science and Information Technologies”
Department of Informatics and Telecommunications
National and Kapodistrian University of Athens

2023-24



Special Values I

NA (not available)

A placeholder for a missing value

```
> x <- c(1, NA, 10, 100)
> x
[1] 1 NA 10 100
> mean(x)
[1] NA
> mean(x, na.rm=T)
[1] 37
```



Special Values II

NULL

NULL represents the absence of a value, as opposed to the NA representing the lack of an existing value

```
> x <- c(1, NULL, 10, 100)
> x
[1] 1  NULL  10  100
> mean(x)
[1] 37
```



which()

```
> x = 10:20
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> x%%2==1
```



which()

```
> x = 10:20
> x
[1] 10 11 12 13 14 15 16 17 18 19 20
> x%%2==1
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
     TRUE FALSE
> which(x%%2==1)
[1]  2  4  6  8 10
```



which()

```
> x = 10:20
> x
[1] 10 11 12 13 14 15 16 17 18 19 20
> x%%2==1
[1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE
     TRUE FALSE
> which(x%%2==1)
[1] 2 4 6 8 10
> x[which(x%%2==1)]
[1] 11 13 15 17 19
```



subset()

```
> x = 10:20
> x
[1] 10 11 12 13 14 15 16 17 18 19 20
> subset(x, x > 13)
[1] 14 15 16 17 18 19 20
```



length()

```
> x <- c(1,2,3)
> length(x)
[1] 3
> x=NULL
> length(x)
[1] 0
> x=NA
> length(x)
[1] 1
```



dim()

```
> a <- array(1:6, dim=c(3,2))
> a
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> dim(a)
[1] 3 2
> length(a)
[1] 6
> attributes(a)
$dim
[1] 3 2
```



colnames()/rownames

```
> a <- array(1:6,dim=c(3,2))
> a
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> colnames(a)
NULL
> rownames(a)=c("a","b","c")
> colnames(a)=c("A","B")
> a
  A B
a 1 4
b 2 5
c 3 6
```



rbind()

```
> a=1:5
> a
 [1]  1  2  3  4  5
> b=6:10
> b
 [1]  6  7  8  9 10
> rbind(a,b)
  [,1] [,2] [,3] [,4] [,5]
a    1    2    3    4    5
b    6    7    8    9   10
```



cbind()

```
> a=1:5
> a
 [1]  1  2  3  4  5
> b=6:10
> b
 [1]  6  7  8  9 10
> cbind(a,b)
      a  b
[1,] 1  6
[2,] 2  7
[3,] 3  8
[4,] 4  9
[5,] 5 10
```



table()

```
> a=c(1,0,1,0,1,0,2,3,4,"hello")
```

```
> table(a)
```

```
a
```

	0	1	2	3	4	hello
a	3	3	1	1	1	1



Set comparison

```
> A=10:19
> B=5:15
> A%in%B
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
     FALSE
> A[A%in%B]
[1] 10 11 12 13 14 15
> A[!A%in%B]
[1] 16 17 18 19
```



Logical operations

Operator	Description
-----------------	--------------------

<code>x&&y</code>	Logical AND (evaluates left to right)
<code>x&y</code>	Logical AND (vectorized - returns vector)
<code>x y</code>	Logical OR (evaluates left to right)
<code>x y</code>	Logical OR (vectorized - returns vector)
<code>!x</code>	Logical NOT

```
> x=c(T,F,F)
> y=c(T,T,T)
> x
[1] TRUE FALSE FALSE
> x&y
[1] TRUE FALSE FALSE
> x[1]&&y[2]
[1] TRUE
```



if/else

```
> a=10
> if (a>5)
+ a=11
> a
[1] 11
> if (a<5)
+ {
+ b=1
+ } else {
+ b=2
+ }
> b
[1] 2
```



loops

```
> for (i in 1:5)
+ print(i)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> for (i in c(2,4))
+ print(i^3)
[1] 8
[1] 64
```



apply() I

```
> a <- array(1:6,dim=c(3,2))
> a
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
>for (i in 1:dim(a)[1])
+ print(mean(a[i,]))
[1] 2.5
[1] 3.5
[1] 4.5
> apply(a,1,mean)
[1] 2.5 3.5 4.5
```



apply() II

```
> a
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> for (i in 1:dim(a)[2])
+ print(mean(a[,i]))
[1] 2
[1] 5
> apply(a,2,mean)
[1] 2 5
```



apply() III

`apply(m, dimcode, f, fargs)`

- m is an array
- $dimcode$ is the dimension of interest
 - 1 indicates rows
 - 2 indicates columns
- f is the function to be applied
- $fargs$ optional arguments to function f



lapply()

```
> mylist=list(1:3,6:7)
> mylist
[[1]]
[1] 1 2 3

[[2]]
[1] 6 7

> lapply(mylist,median)
[[1]]
[1] 2
[[2]]
[1] 6.5
```



sapply()

```
> mylist=list(1:3,6:7)
> mylist
[[1]]
[1] 1 2 3

[[2]]
[1] 6 7

> sapply(mylist,median)
[1] 2.0 6.5
```



Where am I?

- Current location:

```
> getwd()  
[1] "/home/alexdem"
```

- Change location:

```
> setwd("/usr")  
> getwd()  
[1] "/usr"
```

- List directory contents:

```
> list.files()  
[1] "bin" "games" "Gemote" "include"  
[5] "lib" "lib32" "libx32" "local"  
[9] "mips-linux-gnu" "sbin" "share" "src"
```



Read from a text file I

read.table()

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file

Indicative parameters:

- *file*: the name of the file which the data are to be read from
- *header*: logical value. Is the first line a header?
- *sep*: the field separator character
- *quote*: the set of quoting characters
- *as.is*: the default behavior of read.table is to convert character variables (which are not converted to logical, numeric or complex) to factors



Read from a text file II

- *strip.white*: logical. Allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped)
- ...

Similar functions: `read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()`



Read from a text file III

```
alexdem@pine:$ cat file1.tsv
```

```
Name      ID
alex      1234
john      4567
mary      1097
```

```
> read.table("file1.tsv")
```

```
  V1  V2
1 Name  ID
2 alex 1234
3 john 4567
4 mary 1097
```



Read from a text file IV

```
> read.table("file1.tsv",header=T)
```

```
  Name    ID  
1 alex 1234  
2 john 4567  
3 mary 1097
```



Attention!

- The path must be inside double quotes
- In Linux & Mac OS the path has the form /tmp
- In Windows the path has the form c:\\tmp
- Spaces in the file header (usually) create reading problems
- The wrong *sep* choice definitely leads to reading errors!



Writing to a text file I

write.table()

write.table prints its required argument x (after converting it to a data frame if it is not one nor a matrix) to a file or connection

Similar arguments to read.table()

```
> a=array(1:12,dim=c(3,4))
> a
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>write.table(a,"/tmp/file2.txt")
```



Writing to a text file II

```
alexdem@pine:~$ cat /tmp/file2.txt  
"V1" "V2" "V3" "V4"  
"1" 1 4 7 10  
"2" 2 5 8 11  
"3" 3 6 9 12
```



Writing to a text file III

```
write.table(a, "/tmp/file2.txt", quote = F, row.names =  
            F, col.names = F)
```

```
alexdem@pine:~$ cat /tmp/file2.txt  
1 4 7 10  
2 5 8 11  
3 6 9 12
```



Excel files

External packages, such as `readxl`, `XLConnect`, `xlsx`, `gdata`, ... allow reading from Excel files but also **writing** to them. E.g.:

```
library(readxl)
A=read_xlsx("/tmp/file2.xlsx",sheet = 1)
library("xlsx")
write.xlsx(x = A, file = "/tmp/file3.xlsx", sheetName =
  "Status", row.names = FALSE)
```



External packages/libraries

- Plenty of available packages
- They can be loaded to a session environment using the commands `library()` or `require()`, e.g. `require("readxl")`
- An existing package can easily be downloaded and installed from a repository, e.g.
`install.packages("MASS")`
- For bioinformatic applications, the largest repository is bioconductor (<https://www.bioconductor.org/>)

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install()
BiocManager::install(c("GenomicFeatures",
  "AnnotationDbi"))
```



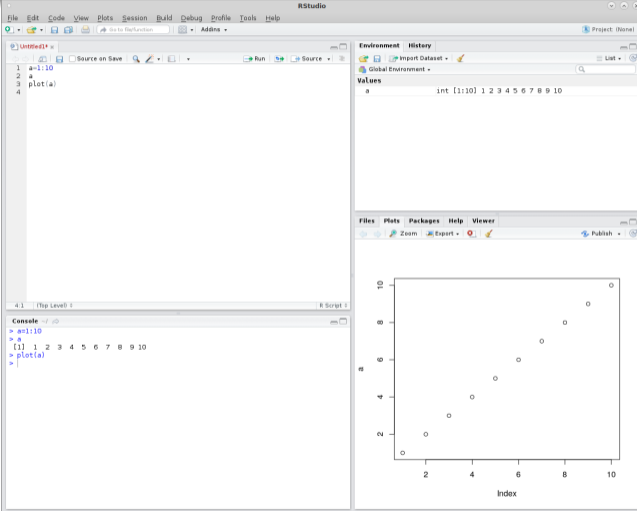
Rstudio IDE I

Rstudio (<https://www.rstudio.com/products/rstudio/>)

- An integrated development environment (IDE) for R
- It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.
- Available in open source and commercial editions
- Runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).



Rstudio IDE II



The screenshot displays the RStudio IDE interface. The main editor window shows a script with the following code:

```
1 a=1:10
2 a
3 plot(a)
4
```

The console window shows the execution output:

```
> a=1:10
> a
[1] 1 2 3 4 5 6 7 8 9 10
> plot(a)
>
```

The Environment pane shows the variable 'a' with the value 'int [1:10] 1 2 3 4 5 6 7 8 9 10'.

The Plots pane shows a scatter plot of the variable 'a'. The x-axis is labeled 'Index' and ranges from 1 to 10. The y-axis is labeled 'a' and ranges from 1 to 10. The plot displays a series of points representing the values of 'a' for each index.

Index	a
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10



Questions?

