LECTURE 2

Matrices

```
m <- matrix(c(1:6), nrow = 2)
m
```

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

```
m1 <- matrix(c(1:6), ncol=2)
m1
```

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

```
m2 <- matrix(c(1:6), nrow=2, byrow=T)
m2
```

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

```
m3 <- matrix(c(1:6), nrow=2, byrow=F)
m3
```

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

```
class(m)
```

```
## [1] "matrix" "array"
```

```
dim(m)
```

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

Apply

To apply the same function to all rows (MARGIN = 1) of a matrix:

```
mean_m <- apply(X=m, MARGIN=1, FUN=mean)
mean_m
```

```
## [1] 3 4
```

... or the all columns (MARGIN = 2):

```
mean_m <- apply(X=m, MARGIN=2, FUN=mean)
mean_m
```

```
## [1] 1.5 3.5 5.5
```

```
class(mean_m)
```

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

```
length(mean_m)
```

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

`mean_m` is a numeric vector.

Named vectors

You can assign names after having intitialized your vector:

```
x <- 1:3
names(x) <- c("One", "Two", "Three")
x
```

```
##   One   Two Three
##     1     2     3
```

...or while you're initializing the vector:

```
x <- c(One = 1, Two = 2, Three = 3)
x
```

```
##   One   Two Three
##     1     2     3
```

So now we can extract element from the vector also calling them by their name instead of the index:

```
x[1] #by index
```

```
## One
##   1
```

```
x[c(1,3)]
```

```
##   One Three
##     1     3
```

```r
x["One"] #by name
```

```
## One
##   1
```

```r
x[c("One", "Three")]
```

```
##   One Three
##     1     3
```

Lists

While vectors can only contain objects of the same class, lists can host elements from different classes (types).

```r
x <- list(1, "cat", TRUE)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "cat"
##
## [[3]]
## [1] TRUE
```

```r
class(x)
```

```
## [1] "list"
```

```r
a <- matrix(c(1:6), nrow=2)
y <- list(matrix=a, description="This is a matrix", is_integer=TRUE)
y
```

```
## $matrix
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## $description
## [1] "This is a matrix"
##
## $is_integer
## [1] TRUE
```

```r
length(y)
```

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

```
names(y)
```

```
## [1] "matrix"      "description" "is_integer"
```

Elements from a list are extracetd using "[[]]" either by index or name:

```
y[[1]]
```

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

```
y[["matrix"]]
```

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

```
class(y[[1]])
```

```
## [1] "matrix" "array"
```

Another, more direct way to extract elements from a list is the following:

```
y$matrix
```

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

```
y$description
```

```
## [1] "This is a matrix"
```

```
y$is_integer
```

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

Subsetting a list

Use the [] (that differently from [[]] do not extract elements from a list):

```
sub <- y[1:2]
sub
```

```
## $matrix
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## $description
## [1] "This is a matrix"
```

```
class(sub)
```

```
## [1] "list"
```

```
sub1 <- y[3]
sub1
```

```
## $is_integer
## [1] TRUE
```

```
class(sub1)
```

```
## [1] "list"
```

Functions returning complex objects

```
x <- c(0.4,0.5,0.3,0.6)
y <- c(1.1, 1.0,1.2,0.9,1.1)
test <- t.test(x,y)
class(test)
```

```
## [1] "htest"
```

```
length(test)
```

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

```
names(test)
```

```
##  [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
##  [6] "null.value"  "stderr"      "alternative" "method"      "data.name"
```

```
test
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -7.4155, df = 6.1238, p-value = 0.000281
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  -0.8103002 -0.4096998
## sample estimates:
## mean of x mean of y
##      0.45      1.06
```

You can extract elements from this object as you can do for lists:

```
test$statistic
```

```
##         t
## -7.415534
```

```
class(test$statistic)
```

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

"statistic" contains the t value (t), "parameter" contains parameters of the distribution used to compute the P-value (df), "p.value" contains the P-value. If we are just interested in the P-value we could also put it in a variable called `p`: `p <- test[["p.value"]]` or `p <- test$p.value`

Writing a function to normalize RNA-seq counts

This function returns the $log(x + a, base =)$ for specified values of x and base:

```
logtransf <- function(x, base=2, a=1) {
  log(x+a, base=base)
}

# 'x' is mandatory. 'base' and 'a' have default values when specific ones are not provided.

f <- logtransf(x=3)
f
```

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

.. which is indeed the output of $log_2(3 + 1)$. Note that if you intend to keep the default 'b' and 'a' value, you don't have to give them as input in when you call the function (as their default values are specified in the script, when the function is written).

Practical example of logarithmic transformation of read coounts from an RNA-seq experiment

We need to transform the mean expression of each gene (=each row):

```
setwd("C:/Users/seren/Desktop/Provero")
load("./count.RData")
x <- apply(X=count, MARGIN = 1, FUN = mean)
head(x)
```

```
##       TSPAN6         TNMD         DPM1        SCYL3     C1orf112          FGR
## 2.654167e+01 2.083333e-02 4.401875e+02 7.604375e+02 1.983333e+02 3.644875e+03
```

... and the logarithmic transformation is $log_2(x + 0.01)$. We do this:

```
logtransf <- function(x, base = 2, a = 0.01) {
  log(x+a, base=base)
}

transf <- logtransf(x=count, base=2, a=0.01)
```

The R environment

Where the object you create (vectors, matrices, lists, functions) appear. With `ls()` you create a character vector containing all the objects inside your environment. To terminate an R session, but keeping the objects, type:

```
save.image(file = "filename.RData")
```

To load a set of objects that you saved in the past, type:

```
load("filename.RData")
```

Saving objects into .RData files allows to exchange data between different R sessions.

To remove more than one object, type:

```
rm(list=c("obj1", "obj3"))
```

```
## Warning in rm(list = c("obj1", "obj3")): object 'obj1' not found
```

```
## Warning in rm(list = c("obj1", "obj3")): object 'obj3' not found
```

To remove all the objects:

```
rm(list = ls())
```

Working directory

```
getwd() # tells where you are located
```

```
## [1] "C:/Users/seren/Desktop/md files/.Rmd files"
```

```
setwd("C:/Users/seren/Desktop") # moves you to the Desktop directory
```