# script_1

## Chapter 1

### R as a calculator

R can be used as a calculator. For example, to calculate the square root of the absolute difference between $3^2$ and $4^2$, you can use the following code:

```
3^2
exp(2)
log(10)
log10(10)
sqrt(abs(3^2-4^2))
```

### Objects in R

You can assign values to objects in R using `<-` or `=` and calculate with them like with numbers. For example:

```
test <- 56
test2 <- 124251762

test + test2
test * test2
```

Objects in workspace can be listed with `ls()` and removed with `rm()`. For example:

```
ls()
rm(test)
```

### Exercise

Solutions to exercises are as follows

```
3^2
sqrt(9)
pi^2
sqrt(abs(3^2-4^2))

test <- 3^2
test2 <- 4^2
test3 <- abs(test - test2)
sqrt(test3)

log(exp(4))
log(exp(4), base = exp(1))
log(100, base = 10)

factorial(8)
exp(factorial(3))
```

## Chapter 2

### Functions in R

Functions are called by passing arguments to them. For example:

```
mean(5)
mean(x = 5)
mean(y = 5)
mean(x="test")
mean(x = test)
```

Functions can also be defined by the user using the `function` keyword. For example:

```
foo <- function(x, y){
  x^2+y^2
}
foo(y=3, x=5)
```

Note that functions should only use the arguments that are defined in the arguments section. Avoid using global variables in functions like this

```
foo <- function(x, y){
  x^2+y^2+ test
}
foo(y=3, x=5)
```

### Exercises

1. Formulate the EOQ formula in R

```
cost_eoq_fun <- function(q, d, co, cl) {
  # returns total cost per period
  # d...demand
  # q...lot size
  # co...ordering cost
  # cl...stock holding cost
  ((1/2)*cl*q)+((d/q)*co)
}

# test cost function
cost_eoq_fun(d=100, q=20, cl = .1, co = 50)
## [1] 251

eoq_fun <- function(co, d, cl) {
  # return optimal lot size
  # d...demand
  # co...ordering cost
  # cl...stock holding cost
  sqrt((2*co*d)/cl)
}
# optimal lot size
q.star <-  eoq_fun(d = 100, cl = .1, co = 50)
# optimal cost
cost_eoq_fun(d=100, q=q.star, cl = .1, co = 50)
## [1] 31.62278
```

2. Derive a function for calculating weighted Euclidean distance between two points.

```r
weuc_d2_func <- function(x,y,w) {
  # calculates weighted Euclidean distance between x and y
  # y,x...vectors
  # w.. weight vector
  sqrt(sum(w*(x-y)^2))
}
# test distacne function
weuc_d2_func(x = c(1,2,3), y= c(3,2,1), w=c(1,1,1) )
## [1] 2.828427
# result should be sqrt(8)
```

Alternatively use intermediate variables within the function

```r
weuc_d2_func <- function(x,y,w) {
  # calculates weighted Euclidean distance between x and y
  # y,x...vectors
  # w.. weight vector
  tmp.diff <- (x-y)
  tmp.diff.sq <- tmp.diff^2
  tmp.result <- w*sqrt(tmp.diff.sq)
  return(tmp.result)
}
# test distacne function
weuc_d2_func(x = c(1,2,3), y= c(3,2,1), w=c(1,1,1) )
## [1] 2 0 2
# result should be sqrt(8)
```

3. Formulate a function for the Geometric Poisson density distribution.

```r
geom_pois_dens_fun <- function(n, lambda, theta){
  # calculates density value of geometric Poisson distribution
  # n...integer, demand/successes, theta,lambda..parameters
  k.vec <- 1:n
  sum(exp(-lambda)*lambda^k.vec/factorial(k.vec)*(1-theta)^(n-k.vec)*choose(n-1, k.vec-1))
}
# test function
geom_pois_dens_fun(n=3, lambda=.5, theta = 2)
## [1] 0.1642687
```

# Chapter 3

Basics on data types and data manipulation

## Create vectors

There are numerous ways to create vectors in R. For example, you can use the `c()` function to concatenate elements into a vector:

```
x <- c(1,2,3,4,5)
# not working as expected because different data types
y <- c(1,2,3,5, "test")
y <- 1:5
rep(x = y, times = 10)
x <- rep(x = y,  each = 10)
z <- c(y,x)
zz <- seq(from = 10, to = 35, by =.25)
```

## Create matrices

Matrices can be created by combining vectors or using `matrix()`. For example:

```
# matrix with numbers
A <- matrix(1:9, ncol=3)
# matrix with characters
B <- matrix(letters[1:9], ncol=3)
# matrix by column-binding
C <- cbind(c(1,2,3,4,5), c(6,7,8,9,10))
# for data frames similar
df.1 <-  data.frame(zz[1:30], as.character(zz[1:30]))
```

## Indexing vectors and matrices

Vectors and matrices can be indexed using square brackets. For example:

```
x <- 1:10
# first element
x[1]
# first five elements
x[1:5]
```

```r
# last element
x[-1]
# elements greater than 5
x[x>5]

A <- matrix(1:9, ncol=3)
# single element
A[1,2]
# first row
A[1,]
# first column
A[,1]
# first two rows
A[1:2,]
```

### Biathlon data set

The biathlon data set should be imported by “import dataset” menu. Once imported as object `biathlon_results_women` you can access it as follows:

```r
# View DataFrame
View(biathlon_results_women)

# overview on race types
table(biathlon_results_women$type)

# overview on competition types
table(biathlon_results_women$competition)

# filter for sprint races only
tmp.id.S <- biathlon_results_women$competition == "S"
df.sprint <- biathlon_results_women[tmp.id.S, ]
# mean of total tiem
mean(df.sprint$tot.time)

# filter with tidyverse
library(tidyverse)
biathlon_results_women %>%
  filter(competition == "S") %>%
  select(tot.time) %>%
```

```
  unlist() %>%
  mean()
```

### Exercises

1. Calculate the outer product of two vectors (without outer())

```
x <- 1:5
y <- 10:6
as.matrix(x) %*% t(as.matrix(y))
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   10    9    8    7    6
## [2,]   20   18   16   14   12
## [3,]   30   27   24   21   18
## [4,]   40   36   32   28   24
## [5,]   50   45   40   35   30
outer(x,y)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   10    9    8    7    6
## [2,]   20   18   16   14   12
## [3,]   30   27   24   21   18
## [4,]   40   36   32   28   24
## [5,]   50   45   40   35   30
```

2. Define a function that calculates the trace of a matrix.

```
trace_func <- function(z){
  # calculates trace of z
  # z...matrix
  sum(diag(z))
}
tmp <- rnorm(9)
A <- matrix(tmp, ncol=3, byrow = T)
trace_func(A)
## [1] -1.142015
```

3. Create a vector containing the first 100 Fibonacci numbers. Most commonly, the Fibonacci numbers are defined recursively by $F_n = F_{n-1} + F_{n-2}$ whereby $F_0 = 0$ and $F_1 = 1$. However, there is also an explicit formulation: $F_n = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$ (check here)

```r
fib_num_fun <- function(n){
  # calculate nth Fibonacci number
  # n...number
  k.vec <- 0:floor((n-1)/2)
  sum(choose(n-k.vec-1, k.vec))
}
# vectorize fib_num_fun such that it accepts input vectors
vfib_num_fun <- Vectorize(fib_num_fun)
# doesn't work
fib_num_fun(1:10)
## Warning in 0:floor((n - 1)/2): numerical expression has 10 elements: only the
## first used
## [1] 10
# works
vfib_num_fun(1:100)
##   [1] 1.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 5.000000e+00
##   [6] 8.000000e+00 1.300000e+01 2.100000e+01 3.400000e+01 5.500000e+01
##  [11] 8.900000e+01 1.440000e+02 2.330000e+02 3.770000e+02 6.100000e+02
##  [16] 9.870000e+02 1.597000e+03 2.584000e+03 4.181000e+03 6.765000e+03
##  [21] 1.094600e+04 1.771100e+04 2.865700e+04 4.636800e+04 7.502500e+04
##  [26] 1.213930e+05 1.964180e+05 3.178110e+05 5.142290e+05 8.320400e+05
##  [31] 1.346269e+06 2.178309e+06 3.524578e+06 5.702887e+06 9.227465e+06
##  [36] 1.493035e+07 2.415782e+07 3.908817e+07 6.324599e+07 1.023342e+08
##  [41] 1.655801e+08 2.679143e+08 4.334944e+08 7.014087e+08 1.134903e+09
##  [46] 1.836312e+09 2.971215e+09 4.807527e+09 7.778742e+09 1.258627e+10
##  [51] 2.036501e+10 3.295128e+10 5.331629e+10 8.626757e+10 1.395839e+11
##  [56] 2.258514e+11 3.654353e+11 5.912867e+11 9.567220e+11 1.548009e+12
##  [61] 2.504731e+12 4.052740e+12 6.557470e+12 1.061021e+13 1.716768e+13
##  [66] 2.777789e+13 4.494557e+13 7.272346e+13 1.176690e+14 1.903925e+14
##  [71] 3.080615e+14 4.984540e+14 8.065155e+14 1.304970e+15 2.111485e+15
##  [76] 3.416455e+15 5.527940e+15 8.944394e+15 1.447233e+16 2.341673e+16
##  [81] 3.788906e+16 6.130579e+16 9.919485e+16 1.605006e+17 2.596955e+17
##  [86] 4.201961e+17 6.798916e+17 1.100088e+18 1.779979e+18 2.880067e+18
##  [91] 4.660047e+18 7.540114e+18 1.220016e+19 1.974027e+19 3.194043e+19
##  [96] 5.168071e+19 8.362114e+19 1.353019e+20 2.189230e+20 3.542248e+20
```

4. Create a matrix containing the all binominal coefficients up to $n = 50$

```r
pas <- outer(1:10, 1:10, choose)
```

5. Create a list containing – vector of 5 small letters – vector of 5 capital letters – vector of 5 random numbers

```r
l1 <- list(small = letters[1:5], capital = LETTERS[1:5], random = rnorm(5))
```

6. Create a matrix with dimension 4×4 and fill it with random numbers.

```r
# set random seed
set.seed(123)
A <- matrix(rnorm(16), ncol=4)
```

7. For the matrix generated in task 6, check whether its invertable.

```r
det(A) != 0
## [1] TRUE
```

8. For the matrix generated in task 6, check whether it is symmetric.

```r
A == t(A)
##       [,1]  [,2]  [,3]  [,4]
## [1,]  TRUE FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE FALSE
## [3,] FALSE FALSE  TRUE FALSE
## [4,] FALSE FALSE FALSE  TRUE
```