R Notebook Biological Modelling

This is the R file I used during the introduction to R session. I hope it will help you to understand things better.

Let's start with setting up a working directory:

```
setwd("//soliscom.uu.nl/uu/Users/Kesmi101/BiologicalModeling")
```

You can use R as a calculator.

```
10^2+36
```

```
## [1] 136
```

Question: Calculate the percentage of your life you spent in this university:

```
100*(2018-2000)/(2018-1969)
```

```
## [1] 36.73469
```

If I need to use my age again, I have to re-calculate it. It is not very handy. Let me save it in a variable.

```
age<-2018-1969
uuyears<-2018-2000
100*uuyears/age
```

```
## [1] 36.73469
```

Can I use this code next year? Better to put 2018 also as a variable:

```
year<-2018
age<-year-1969
uuyears<- year-2000
age
```

```
## [1] 49
```

```
100*uuyears/age
```

```
## [1] 36.73469
```

```
year<-year+1
year
```

```
## [1] 2019
```

```
age<-year-1969
age
```

```
## [1] 50
```

```
uuyears<- year-2000
100*uuyears/age
```

```
## [1] 38
```

```
d <- 5
d <- d ^ d
print (d)
```

```
## [1] 3125
```

Removing variables from R studio: use the interface.

You can save your scripts, or use somebody else's script in your own R studio session.

Let's go to Biological modelling page and download a script.

Everything in R is a vector, or a high dimensional vector in R. c() function makes vectors.

```
e <- c(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
#But this is not so nice code, you can also do this.
e<- c(10:1)
#You can even take out c()
e<- 10:1
```

If we want to compute the average of all the numbers in variable e:

```
(10+9+8+7+6+5+4+3+2+1)
```

```
## [1] 55
```

```
(10+9+8+7+6+5+4+3+2+1)/10
```

```
## [1] 5.5
```

```
#This is again not so nice!
#Let's use a function instead
sum(e)
```

```
## [1] 55
```

```
sum(e)/10
```

```
## [1] 5.5
```

I can even make it more general: length() function prints the length of a vector. The vectors can contain any type of variables, numbers, characters, etc.

```
length(e)
```

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

```
sum(e)/length(e)
```

```
## [1] 5.5
```

How to get help in R?

1. Tab key helps you to see which functions are available, and which parameters you can use
2. On R studio screen
3. in the console:

```
help(rnorm)
```

```
## starting httpd help server ... done
```

We can have strings in a vector

```
mystrings <- c("I", "love", "R")
print (mystrings)
```

```
## [1] "I"    "love" "R"
```

```
print ( length(mystrings))
```

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

How do we adress the values in a vector?

```
vec1<- c(4, 5, 2 , 6)
vec1
```

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

```
vec1[3]
```

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

```
vec2 <- c(1:4)
vec2
```

```
## [1] 1 2 3 4
```

```
vec1+vec2
```

```
## [1]  5  7  5 10
```

```
sum(vec1)
```

```
## [1] 17
```

```
sum(vec2)
```

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

```
sum(vec1) + sum(vec2)
```

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

Matrices: Another much used data structure.

```
Mymat<- matrix(data=c(1:10), ncol=2)
Mymat
```

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

```
#Adressing values in a matrix:
Mymat[4,2]
```

```
## [1] 9
```

```
#Address a full column
Mymat[,1]
```

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

```
#Address a full row
Mymat[1,]
```

```
## [1] 1 6
```

Often each column in a data matrix gives a different measure belonging to the data point. Therefore, it is handy to have column names. We can have column names in R structure data.frame.

```
data<- data.frame(Time= Mymat[,1], Price=Mymat[,2])
data
```

```
##   Time Price
## 1    1     6
## 2    2     7
## 3    3     8
## 4    4     9
## 5    5    10
```

```
data$Time
```

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

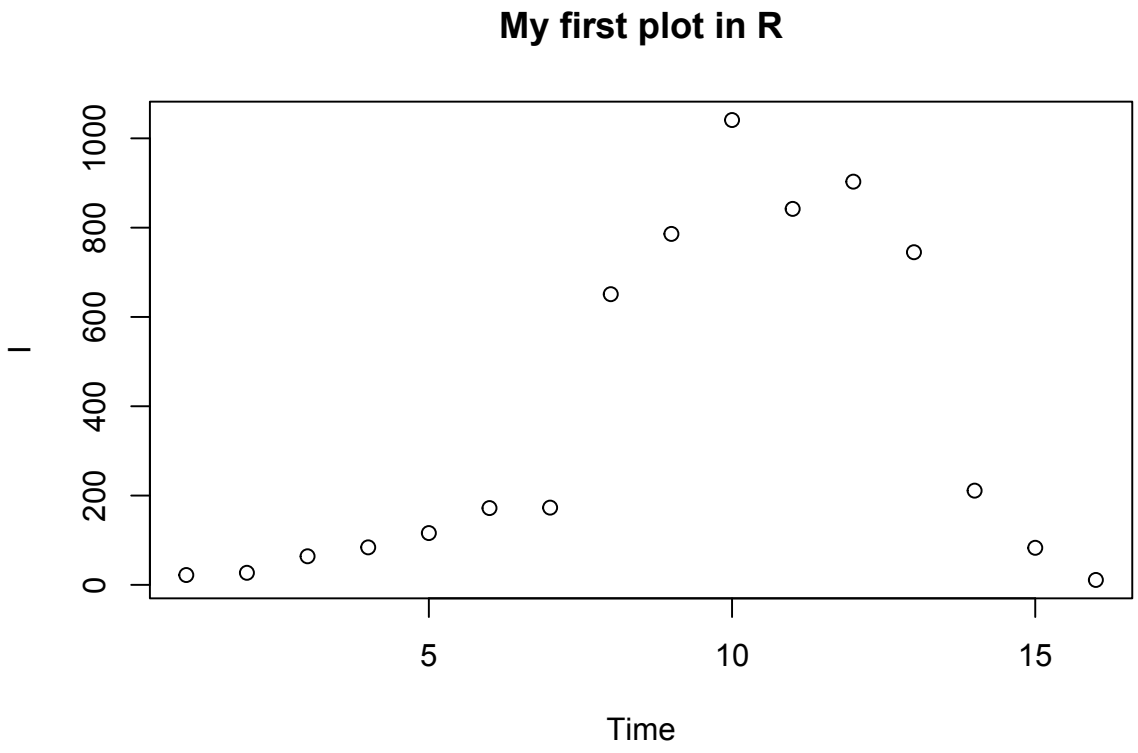
Let's read a data file

```
timedata<-read.table("data/niameyA.txt", header=TRUE)
timedata
```

```
##    time    I
## 1     1   22
## 2     2   27
## 3     3   64
## 4     4   84
## 5     5  116
## 6     6  172
## 7     7  173
## 8     8  651
## 9     9  786
## 10   10 1041
## 11   11  842
## 12   12  903
## 13   13  745
## 14   14  211
## 15   15   83
## 16   16   11
```

Let's plot what we have read:

```
plot(timedata$time, timedata$I, ylab="I", xlab="Time", main="My first plot in R")
```



As we read files, we can also write data to a file:

```
timedata$time <- timedata$time*2
timedata
```

```
##    time    I
## 1     2   22
## 2     4   27
## 3     6   64
## 4     8   84
## 5    10  116
## 6    12  172
## 7    14  173
## 8    16  651
## 9    18  786
## 10   20 1041
## 11   22  842
## 12   24  903
## 13   26  745
## 14   28  211
## 15   30   83
## 16   32   11
```

```
write.table(timedata, file="timedata.txt", quote=FALSE, row.names = FALSE)
```

Functions:

Let's write a function that takes two arguments, and returns the product of two arguments:

```
myprod<- function(arg1, arg2) {
  return (arg1*arg2)

}

myprod(3,5)
```

```
## [1] 15
```

```
firstfun <- function(name) {
  cat("Hello ")
  cat(name)
}
firstfun("Bas")
```

```
## Hello Bas
```

Logical values (TRUE or FALSE) is very important in R. You can use them to select the data you want to have. Here we will look into few operators that return logical values: - “>” and “<” larger - “==” is equal? Take care: it is different than “=” - “!=” not equal

```
h <- 0
phorty_phour <- 44
h > phorty_phour
```

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

```
h< phorty_phour
```

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

```
h == phorty_phour
```

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

```
h != phorty_phour
```

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

You can use the logical values in if() functions to select what you want. If() functions can be embedded in anywhere in your code: in a for loop, in a function etc. We have already used if() function above.

```
i <- 0:6
i[4]
```

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

```
if (i[4]>0) {
  j<- i[4] }else{
    j <- 0}
j
```

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

For loops are difficult to understand at first. But once you understand them, you will see that they are very powerfull. Let’s go step by step with this code.

```
f <- 0
for (kk in 1:5) { # kk is our counter, it can have any name, xx, count, etc.
f <- f + kk
```

```
print (c(kk, f))
}
```

```
## [1] 1 1
## [1] 2 3
## [1] 3 6
## [1]  4 10
## [1]  5 15
```

```
h<-1:8
h
```

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

```
s<-c()
for (i in h)
{s[i] = h[i]*10}
s
```

```
## [1] 10 20 30 40 50 60 70 80
```