Homework 3

Programs must be written for people to read,
and only incidentally for machines to execute.

Hal Abelson, Structure and Interpretation of Computer Programs

Always code as if the guy who ends up maintaining your code
will be a violent psychopath who knows where you live.

John Woods

Readability counts! Please, take that into consideration when you write your code.

1. Iris

Assume you like to collect Iris flowers. You have built a table where you write some of the flowers' properties, then you fill in the value of each property, and you do this for every flower you have collected. Here is an excerpt of your table:

Iris sibirica (source: wikimedia.org).

Sample no.	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
1	5.1	3.5	1.4	0.2	Setosa
2	7.0	3.2	4.7	1.4	Versicolor
3	6.3	3.3	6.0	2.5	Virginica

You went to the field and you picked up a new sample. Your new sample has the following properties:

Sample no.	Sepal Length	Sepal Width	Petal Length	Petal Width
New sample	4.7	3.2	1.7	0.3

The problem is you cannot find to which species it belongs. Well, luckily you have some programming skills in R! Use them!

This is a typical classification[1] problem. You will learn more about that in another lecture. For now you can solve this problem with a simple implementation of the *k*-nearest neighbours algorithm[2]:

1 https://en.wikipedia.org/wiki/Statistical_classification
2 https://en.wikipedia.org/wiki/K-nearest_neighbours_algorithm

- Consider each property have the same weight;
- You can see each one of them as an axis, so it is like you have four dimensions;
- The new sample will have the same species as the nearest neighbour (k=1) in your data set;
- Consider that distance as the Euclidean distance[3].

The whole data set is already included in R, you just have to load it with **data** command:

```
> data("iris")
> iris[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
```

Solve this problem in two ways:

a) using a for loop.
b) applying a function over the data set.

2. Easy ones again

In **Homework 1 Exercise 2**, you implemented some functions in R to find the maximum, minimum and average values of a given vector.

Write those functions again, but do not use a for loop (well, if you wrote them without a for loop in the first place, congratulations!). Also, define an optional argument **first_n**, where the user can choose to calculate those values only for the first *n* elements of the given vector.

Try your new geeky and faster functions with the **iris** data set, find out the maximum, minimum and average values of each property

3. Game of Life

The Game of Life[4] is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is the best-known example of a cellular automaton.

The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves or, for advanced players, by creating patterns with particular properties.

In this exercise, you don't have to code. On the other hand, you will try your best to understand and explain a given code snippet (see the last page). The following code already has some comments to help you out. Make a pause and take a snack. Copy the code and run it on your computer. Then study the code for a while. You will see that it is not that complex, and now that you know about functions, your task will be easier.

3 https://en.wikipedia.org/wiki/Euclidean_distance
4 http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

```
# Generates a new board - either a random one, sample blinker or gliders, or user specified.
gen.board <- function(type="random", nrow=3, ncol=3, seeds=NULL)
{
    if(type=="random")
    {
       return(matrix(runif(nrow*ncol) > 0.5, nrow=nrow, ncol=ncol))
    } else if(type=="blinker")
    {
       seeds <- list(c(2,1),c(2,2),c(2,3))
    } else if(type=="glider")
    {
       seeds <- list(c(1,2),c(2,3),c(3,1), c(3,2), c(3,3))
    }
    board <- matrix(FALSE, nrow=nrow, ncol=ncol)
    for(k in seq_along(seeds))
    {
      board[seeds[[k]][1],seeds[[k]][2]] <- TRUE
    }
    board
}

# Returns the number of living neighbours to a location
count.neighbours <- function(x,i,j)
{
   sum(x[max(1,i-1):min(nrow(x),i+1),max(1,j-1):min(ncol(x),j+1)]) - x[i,j]
}

# Implements the rulebase
determine.new.state <- function(board, i, j)
{
   N <- count.neighbours(board,i,j)
   (N == 3 || (N ==2 && board[i,j]))
}

# Generates the next interation of the board from the existing one
evolve <- function(board)
{
   newboard <- board
   for(i in seq_len(nrow(board)))
   {
      for(j in seq_len(ncol(board)))
      {
         newboard[i,j] <- determine.new.state(board,i,j)
      }
   }
   newboard
}

# Plays the game.  By default, the board is shown in a plot window, though output to the console if
possible.
game.of.life <- function(board, nsteps=50, timebetweensteps=0.25, graphicaloutput=TRUE)
{
   if(!require(lattice)) stop("lattice package could not be loaded")
   nr <- nrow(board)

   for(i in seq_len(nsteps))
   {
      if(graphicaloutput)
      {
         print(levelplot(t(board[nr:1,]), colorkey=FALSE))
      } else print(board)

      Sys.sleep(timebetweensteps)

      newboard <- evolve(board)

      if(all(newboard==board))
      {
         message("board is static")
         break
      } else if(sum(newboard) < 1)
      {
         message("everything is dead")
         break
      } else board <- newboard
   }
   invisible(board)
}

# Example usage
game.of.life(gen.board("blinker"))
game.of.life(gen.board("glider", 18, 20))
game.of.life(gen.board(, 50, 50))
```