S2.0 Recap last time

1. Using the function seq(), create the following vector: (-3 -46 -89 -132 -175 -218 -261).
2. Create a loop in which the parameter “cowboy_says” takes on the values “hi”, “whoa”, “howdy” and “yee-haa!”. Print both the length of the different values and the content during each iteration of the loop.
3. Expand that loop, such that when the value of “cowboy_says” equals “yee-haa!”, in addition to the other output, R prints “pow! pow! pow!” (reflecting the cowboy shooting his gun ;).

S2.1 Functions

1. Write a function that returns the input value + 1. What happens when you input a vector of numbers? What happens when you give as an input a vector of strings?
2. Write a function that tells if a string is present inside a vector of strings. (E.g. does the vector c("apple", "pear", "orange") contain the value "pear"?)
3. Write a function that takes as an input a vector of numbers, and as an output it tells you which elements are outliers. We will define an outlier as the number that is larger than the mean + standard deviation of the vector, or that is smaller than the mean - standard deviation of the vector.
   Tip: divide this problem into smaller problems. First determine the standard deviation and mean, the boundaries that follow from this (mean+std and mean-std), then look at which elements are above/below the boundary, then make a function out of this ...
   Which are the outliers in the following vector?
   `> c(1.1, 1.03, 8.1, 2.23, 0.56, -0.8, 1.5, -3, 1.3, -0.9, 0.87)`
4. Write a function named oddcount() that counts the odd numbers in a vector of integers (hint: use the modulo operator %% inside the function)
   Tip: again, divide this problem into smaller problems and solve those one at a time!
5. The Fibonacci sequence is the series of numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, … The next number is found by adding up the two numbers before it. Write a function that returns the Nth element of the Fibonacci sequence.
6. Write a function that returns the first N elements of the Fibonacci sequence.
   Note that if an empty vector `some_vector <- c()` *exists, you can always add elements by saying* `some_vector[2] <- 10` *(in this example, the value 10 is assigned to the 2nd item of the vector).*
7. Another function:
   (a) Write a function that determines the index of the first value that equals 1 in the function’s vector input argument.
   (b) Now adjust the input arguments of that function such that per default it works like described in (a), except that by adjusting an optional input argument you can search for other values than 1.

S2.2: Matrices

1. The following code will generate a 4 x 7 matrix filled with random numbers:
   `my_fake_data_matrix <- matrix(runif(4*7), ncol=7)`
   (a) Select the 3rd column of this matrix.
   (b) Select the first and 3rd column of this matrix.
   (c) Do this again using a logical selection vector (i.e. with TRUE and FALSE).
   (d) Subselect only those elements that are in the 1st and 2nd row, and also in column 4-6.
   (e) When can pretend that this matrix contains actual data and that the vector of strings
   `sample_conditions = c('control', 'control', 'treatment1', 'treatment1', 'treatment2', 'treatment2', 'treatment1')`
   annotates the conditions, with each of the rows some different parameter that was measured. Columns with the same name represent repetitions of the experiment.
   Now select all columns corresponding to treatment 2 using the above vector `sample_conditions`.
2. Create a 2nd matrix using the following code:
   `matrix(c(7, 0, 1, 9, 1, 3, 1, 8, 9, 9, 2, 5, 6, 2, 3, 2, 8, 7, 0, 5, 5, 6, 3, 3, 4, 10, 0, 2),ncol=7)`
   You can use the function `which()` to find out which elements are equal to 3. The returned index is not super-convenient for humans looking at a matrix. By using an optional argument of the which function, you can tell R to output to output row and column numbers. Use the command "?which" to find out how to do this. Which elements of the matrix, expressed in terms of i,j, with i row and j column, are equal to 3?
3. Naming matrix rows and columns.
   With the function "`colnames()`" you can give names to columns. For example:
   ```
   z <- matrix(c(1:4), nrow = 2)
   colnames(z) <- c('a','b')
   z
   ```
   When you type the command `colnames(z)` R will return the column names of matrix z.
   Compare the output of `z[,1]` or `z[,'a']`. Is there any difference?
   How does the function "`rownames()`" work? And the function "`names()`"?
4. You can also define names when defining a 1-d vector.
   (a) What happens in the following code?
   ```
   veg_prices = c(cauliflower=1.7, broccoli=0.74, cucumber=.65, lettuce=.85)
   my_shopping_list = c('cauliflower', 'cauliflower', 'broccoli')
   veg_prices[my_shopping_list]
   ```

(b) Modify this code so that there's also the price of a carrot (you can make up a number), and then calculate how much it would cost (in total) if I'd buy 2 carrots and a cucumber.

(c) Now modify this code such that it does the same, but uses the following as input vector corresponding to the purchase of 2 carrots and a cucumber:

```
my_shopping_list = c(carrot=2, cucumber=1)
```

This might involve some puzzling :)

5. The following code again generates some fake data:

```
my_fake_data_matrix <- matrix(runif(4*7), ncol=7)
colnames(my_fake_data_matrix) = c('control', 'control',
'treatment1', 'treatment1', 'treatment2', 'treatment2',
'treatment1')
rownames(my_fake_data_matrix) = c('Heart size', 'Heart rate',
'Heart compaction', 'Mouse size')
```

Now, use the column names of this vector to select treatment #2 data again. (Note: you might run into some unexpected behavior.)

6. The functions `rbind()` and `cbind()` are very convenient to paste together matrices. Example:

```
matrix1 = matrix(runif(10), ncol=5)
matrix2 = matrix(runif(10), ncol=5)
rbind(matrix1, matrix2)
cbind(matrix1, matrix2)
```

(a) Test this with matrices of different dimensions.

(b) Write a function that returns a matrix, where the first column equals the input vector and the second column is the power 2 of each value plus 1. That is, given the following input variable:

```
c(1:8)
```

The output of the function is:

```
> my_function(c(1:8))
     [,1] [,2]
[1,]    1    2
[2,]    2    5
[3,]    3   10
[4,]    4   17
[5,]    5   26
[6,]    6   37
[7,]    7   50
[8,]    8   65
```

7. Generate a matrix where two columns contain uniform random numbers, and two other other gaussian random numbers. (Hint: there are functions that can generate N random numbers following these respective distributions, use Google or other sources to identify them.)

(a) The functions nrow(z), ncol(z) and dim(z) can be used to determine the numbers of rows, columns or both of a matrix z, respectively. Create a loop that goes from

1 to the number of columns of the generated matrix, which then calculates the mean and standard deviation of each of the columns. Save both values to two respective vectors, where each item corresponds to a column from the generated matrix.

(b) Now compute mean and sd for each column using the apply functions.

(c) Now apply the function that you wrote in exercise s2.1.1 to the columns of the generated matrix.

(d) How can you change the mean or standard deviation of the numbers that are put into the matrix? (Note: answers might be different for the two functions generating the two respective distributions.)