S1.1: Data types

1. Explore the output of the following inputs:

```
> 5:8

> 10:1

> seq(from=12, to=30, by=3)

> seq(from=1.1,to=2,length=10)

> rep(8, 4)

> rep(1:3, 2)

> rep(c(15,12,3), each = 2)
```

2. Given the following code:

```
> var1=3
> var2=5
> var3='1'
```

   which operation is valid (and why)?
   a. var1+var2
   b. var1+var3
   c. var2+var2
   d. Var2+var3

3. Create a vector of string characters with the first names of the people in your lab. Then, make R output those names that do have the letter “M” in the name.

4. Vectors in R are essential. Remarkably, a lot of statistical operations are built in so that you can quickly compute the mean, the standard deviation, the median, etc. from a vector.
   a. To compute the mean of a vector v we use:

```
> mean(v)
```

   Create a vector containing all integers from 1 until 100 where the element 2 is missing (that is, create the vector v = (1, 3, 4, 5, …, 100). What is its mean?

   b. To compute the median and the standard deviation of a vector “v” we use:

```
> sd(v)
> median(v)
```

   What is the standard deviation and the mean in the vector generated before?

5. To sum over all elements in a vector we can use:

```
> sum(v)
```

   a. How much is 1+3+5+...+99? (that is, how much is the sum of all odd numbers between 1 and 100?)
   b. How much is 2+4+6+...+68?
   c. How much is 1-2+3-4+…-98+99?

6. With R, it's also easy to obtain the maximum or minimum value of a vector using `max(v)` or `min(v)`, respectively. Create the following vector in R:

```
v <- c(52.9, 14.7, 78, 70.6, 88.7, 81.4, 77.3, 9, 62.2, 64.9,
80.9, 98.3, 56.5, 98.5, 77.2, 70.5, 38.6, 21.7, 10.6, 66.8, 32.7,
23.6, 31.9, 65.6, 32.7, 6.5, 77.7, 74.9, 2.1, 2.8, 78.2, 7.2,
95.6, 78.4, 14.3, 37.1, 59.3, 86.4, 46.6, 16.5, 11.6, 39.6, 18.4,
6, 34.1, 18.6, 47.2, 5.3, 49.9, 91.3, 27.2, 96.9, 8.6, 44.5, 73,
11.3, 66.4, 12.4, 35.1, 20.5, 6, 81.5, 97.6, 26.7, 39.9, 33.6,
88.7, 46.5, 66.8, 66.4, 45.5, 82.1, 37.8, 66.2, 36.3, 57.2, 89.4,
88.4, 7.5, 64.1, 53.8, 91.5, 55.4, 69.6, 3.5, 83.2, 90.5, 36.9,
6.6, 85.5, 24.3, 11.3, 14.3, 6.2, 28.9, 17.5, 25.5, 91.7, 91.6,
57.1)
```

   Now determine the mean, standard deviation, maximum and minimum values of vector "`v`" we just defined.

7. Execute the following code in R:

```
monthly_savings <- c(100, 100, 100, -200, 300, -100,
    100, 100, -200, 100, 100, 100)
```

   Using the function `cumsum(v)` on vector `v`, we can calculate the *cumulative sum*. Try this on the vector "monthly_savings". If "monthly_savings" contains the amounts I added to a special savings account last year, in which month was I able to afford a 500 € new phone?

8. When you use the command "`which(v)`", where v is a logical vector (i.e. containing "`True`" or "`False`" values) it will return the positions of the vector that contain `True` values. In this exercise:
   a. Use the *`monthly_savings`* vector to create a new logical vector of which the items correspond to the monthly_savings vector, with `True` values for positive amounts, and `False` values for negative amounts.
   b. Then use that new vector to select all positive amounts from the `monthly_savings` vector.
   c. Use the which function to identify months in which the savings were positive. (An output would look like, "3, 4, 5" to indicate that March-May had positive savings.

d. What do I get when I type "`monthly_savings[monthly_savings<0]`" in R?

9. Non-intuitive behavior. Be aware R sometimes "helps" you. Kudo's for those who can find out why the output of "`monthly_savings[c(T,T,F)]`" looks like it does.

10. Have R calculate the sum of the squares of the numbers 1 to 10 (this can be done with functionalities we've just talked about).

11. Make R determine which entries equal '`WT`' in the vector "`c('WT', 'WT', 'KO', 'KO', 'WT', 'KO', 'control', 'control')`".

12. We can use an additional functionality of vectors to annotate them. That is to say, we can give each entry/element a name. This can be done in two ways:
(1) Given a vector "`v`" with entries `c(1,2,3)`, type `names(v) <- c('name1', 'name2', 'cookie')`.
(2) When creating the vector, by saying: `v <- c(name1=1, name2=2, cookie=3)`.

    a. Try both approaches in R and inspect how `v` now looks.
    b. Create a vector with values 1 to 5, and give each entry a name.
    c. Values can be selected by their names, try what you get when you type "`v['cookie']`".

14. Conversely, you can retrieve names of a vector by typing `names(v)`. What would you get when you'd type `which(names(v)=='name2')` for the above vector `v`?

S1.2: Loops: for and if

15. Copy the following code to R and change it such that not the value of the variable `idx`, but the square of the value gets printed:

```
for (idx in 1:10) {
    print( idx )
}
```

16. Modify the following code such that the values over which the loop goes (12,3,64,23,12) are summed in the parameter `a`.
That is to say, when the loop execution is done, `a` should have value 114.

```
a = 0
for (value in c(12,3,64,23,12)) {
    a = XXXX
}
a
```

17. The modulo operator, %% in R, returns the residue of the division of one number by another. For example, when we type:

```
> 5%%2
```

We get 1 as an output, indicating that the residue of the division of 5 by 2 is equal to 1. When we type:

```
> 10%%2
```

Then we get 0, since 10%2=5 with no residue. Therefore, we could for instance use %%2 to find which integers are odd and which are even, based on the output of %%.
Define a vector of integer numbers (for instance, `v <- 1:13` or `v<-c(30,21,24,-1,21,-32)`). Write a for loop that prints out the elements of this vector that are odd numbers.

18. Using for loops, if statements, and statistical operators for vectors, write a code in R that performs the following steps:

- For a given vector, it computes the mean increase of sequential elements (i.e. mean of `v[i]-v[i-1]`).
- If the mean increase of sequential elements is positive, zero or negative, then it assigns the value +1, 0 or -1, respectively, to a parameter that we will call `slope`.

19. Write an R code to find out whether the maximum value of a vector is larger than 0.

20. Write an R code that gives the maximum negative value of a vector made of positive and negative values, such as:

```
> v <- c(3,0,-3,1,-31,2,5,-6,10)
```

21. (more advanced)
The Fibonacci series is defined as a series where each element is the sum of the two previous entries. (E.g. 1 1 2 3 5 8 13 21 34.) See also https://en.wikipedia.org/wiki/Fibonacci_number. Change the following code such that it generates the first 10 Fibonacci numbers (let the series start at 1, 1, …, as is already set up below):

```
N=10
fibonacci_series = c(1,1)

for (idx in 2:(N-1)) {
    fibonacci_series[idx+1] = XXXXX
}
print(fibonacci_series)
```

22. (more advanced)
A prime number is defined as a number that can only be divided by itself and 1. (Where “can be divided” here means it gives an integer outcome and no residue.) We can determine if a number is a prime number by “brute force” using a loop. Adapt the following loop so that it loops over an

appropriate range of numbers to test whether `my_number` is a prime. For this test, you can use the modulo operator from the previous assignment, combined with an if-statement.
As you can see, when it is divisible by a number not itself or 1, we can store this in a logical, such that we in the end know whether this number was a prime number or not.
For reference, here are some prime numbers: 1 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71.

```
my_number <- 56
Prime_yes_no <- T
for (XXXX) {

        if (YYYYYY) {
            prime_yes_no <- F
        }

}
prime_yes_no
```