Introduction to R - Part 1

== Installation : TinyTex ==

```
tinytex::install_tinytex(force = TRUE)
```

```
## tlmgr install tlgpg
```

```
## tlmgr update --self
```

```
## tlmgr install tlgpg
```

```
## tlmgr --repository http://www.preining.info/tlgpg/ install tlgpg
```

```
## tlmgr option repository "https://mirror.kku.ac.th/CTAN/systems/texlive/tlnet"
```

```
## tlmgr update --list
```

==== Arithmetic with R ====

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`
- Quotient: `%/%`
- Modulo (Remainder): `%%`

```
# An addition
5 + 5
```

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

```
# A subtraction
5 - 5
```

```
## [1] 0
```

```r
# A multiplication
3 * 5
```

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

```r
# A division
(5 + 5) / 2
```

```
## [1] 5
```

```r
# Exponentiation
2^5
```

```
## [1] 32
```

```r
# Quotient
28%/%6
```

```
## [1] 4
```

```r
# Modulo
28%%6
```

```
## [1] 4
```

==== Variable assignment ====

A basic concept in (statistical) programming is called a variable. A variable allows you to store a value (e.g. 4) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

```r
# Assign a value 69 to a variable my_var
my_var = 69
# Print out the value of the variable x
my_var
```

```
## [1] 69
```

```r
# Assign the value 5 to the variable my_apples and 6 to the variable my_oranges
my_apples <- 5
my_oranges <- 6

# Add these two variables together
my_fruit <- my_apples + my_oranges
# Create the variable my_fruit
my_fruit
```

```
## [1] 11
```

==== Basic data types in R ====

R works with numerous data types. Some of the most basic types to get started are:

- Decimal values like 4.5 are called numerics.
- Whole numbers like 4 are called integers. Integers are also numerics.
- Boolean values (TRUE or FALSE) are called logical.
- Text (or string) values are called characters.

What's that data type? When you add 5 + "six", you will got an error due to a mismatch in data types.You can avoid such embarrassing situations by checking the data type of a variable beforehand. You can do this with the class() function, as the code in the editor shows.

```r
# Assign 42.5 to a variable my_numeric
my_numeric <- 42.5
# Assign 4 to a variable my_integer
my_integer <- 4
# Assign "universe" to a variable my_character
my_character <- "universe"
# Assign FALSE to a variable my_logical
my_logical <- FALSE
# Check class of my_numeric, my_integer, my_character and my_logical
class(my_numeric)
```

```
## [1] "numeric"
```

```r
class(my_integer)
```

```
## [1] "numeric"
```

```r
class(my_character)
```

```
## [1] "character"
```

```r
class(my_logical)
```

```
## [1] "logical"
```

==== Create a vector ====

Let's take a trip to the City of Sins, also known as Statisticians Paradise! You will learn how to uplift your performance at the tables and fire off your career as a professional gambler. This section will show how you can easily keep track of your betting progress and how you can do some simple analyses on past actions. Next stop, VEGAS!!

Vectors are one-dimension arrays that can hold numeric data, character data, or logical data. In other words, a vector is a simple tool to store data. For example, you can store your daily gains and losses in the casinos.

In R, you create a vector with the combine function `c()`. You place the vector elements separated by a comma between the parentheses. For example:

`numeric_vector <- c(1, 10, 49)`

`character_vector <- c("a", "b", "c")`

After one week in Las Vegas and still zero Ferraris in your garage, you decide that it is time to start using your data analytical superpowers. Before doing a first analysis, you decide to first collect all the winnings and losses for the last week:

For `poker_vector`: On Monday you won $140 Tuesday you lost $50 Wednesday you won $20 Thursday you lost $120 Friday you won $240

For `roulette_vector`: On Monday you lost $24 Tuesday you lost $50 Wednesday you won $100 Thursday you lost $350 Friday you won $10

To be able to use this data in R, you decide to create the variables `poker_vector` and `roulette_vector`.

```r
# Assign the value "Go!" to the variable vegas. Remember: R is case sensitive!
vegas <- "Go!"
# Complete the code such that boolean_vector contains the three elements: TRUE, FALSE and TRUE (in that
booleaan_vector <- c(TRUE , FALSE , TRUE)

# Poker winnings from Monday to Friday
poker_vector <- c(140 , -50 , 20 , -120 , 240)
# Roulette winnings from Monday to Friday
roulette_vector <- c(-24 , -50 , 100 , -350 , 10)
```

==== Naming a vector ====

As a data analyst, it is important to have a clear view on the data that you are using. Understanding what each element refers to is therefore essential. You can give a name to the elements of a vector with the `names()` function. Have a look at this example:

`some_vector <- c("John Doe", "poker player")`

`names(some_vector) <- c("Name", "Profession")`

This code first creates a vector `some_vector` and then gives the two elements a name. The first element is assigned the name `Name`, while the second element is labeled `Profession`.

In the previous exercise, we created a vector with your winnings over the week. Each vector element refers to a day of the week but it is hard to tell which element belongs to which day. It would be nice if you could show that in the vector itself.

Here, you probably experienced that it is boring and frustrating to type and retype information such as the days of the week. However, when you look at it from a higher perspective, there is a more efficient way to do this, namely, to assign the days of the week vector to a variable! Just like you did with your poker and roulette returns, you can also create a variable that contains the days of the week. This way you can use and re-use it.

```r
# Assign days as names of poker_vector
names(poker_vector) <- c("Mon" , "Tue" , "Wed" , "Thu" , "Fri")

# Assign days as names of roulette_vector
names(roulette_vector) <- c("Mon" , "Tue" , "Wed" , "Thu" , "Fri")
```

```r
## call poker_vector , roulette_vector
poker_vector
```

```
##  Mon  Tue  Wed  Thu  Fri
##  140  -50   20 -120  240
```

```r
roulette_vector
```

```
##  Mon  Tue  Wed  Thu  Fri
##  -24  -50  100 -350   10
```

```r
# Create a variable days_vector that contains the days of the week.
days_vector <- c("Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday")

# Use days_vector to set the names of poker_vector and roulette_vector
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector
```

==== Calculating total winnings ====

Now that you have the poker and roulette winnings nicely as named vectors, you can start doing some data analytical magic.

You want to find out the following type of information:

- How much has been your overall profit or loss per day of the week?
- Have you lost money over the week in total?
- Are you winning/losing money on poker or on roulette?

To get the answers, you have to do arithmetic calculations on vectors. It is important to know that if you sum two vectors in R, it takes the element-wise sum.

Now you understand how R does arithmetic with vectors, it is time to get those Ferraris in your garage! First, you need to understand what the overall profit or loss per day of the week was. The total daily profit is the sum of the profit/loss you realized on poker per day, and the profit/loss you realized on roulette per day.

Based on the previous analysis, it looks like you had a mix of good and bad days. This is not what your ego expected, and you wonder if there may be a very tiny chance you have lost money over the week in total?

A function that helps you to answer this question is `sum()`. It calculates the sum of all elements of a vector.

Oops, it seems like you are losing money. Time to rethink and adapt your strategy! This will require some deeper analysis.

After a short brainstorm in your hotel's jacuzzi, you realize that a possible explanation might be that your skills in roulette are not as well developed as your skills in poker. So maybe your total gains in poker are higher (or `>`) than in roulette.

```r
A_vector <- c(1, 2, 3)
B_vector <- c(4, 5, 6)
# Take the sum of the variables A_vector and B_vector and assign it to total_vector.

total_vector <- A_vector + B_vector

# Inspect the result by printing out total_vector.
total_vector
```

```
## [1] 5 7 9
```

```r
# Assign to the variable total_daily how much you won or lost on each day in total (poker and roulette
total_daily <- poker_vector + roulette_vector
total_daily
```

```
##    Monday   Tuesday Wednesday  Thursday    Friday
##       116      -100       120      -470       250
```

```r
# Calculate the total amount of money that you have won/lost with poker and assign to the variable tota
total_poker <- sum(poker_vector)

# Calculate the total amount of money that you have won/lost with roulette and assign to the variable t
total_roulette <- sum(roulette_vector)

# Now that you have the totals for roulette and poker, you can easily calculate total_week (which is th
total_week <- sum(total_poker + total_roulette)
# Print out total_week.
total_week
```

```
## [1] -84
```

```r
# Check if you realized higher total gains in poker than in roulette
poker_morethan_roulette <- total_poker > total_roulette
poker_morethan_roulette
```

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

==== Vector selection ====

Another possible route for investigation is your performance at the beginning of the working week compared to the end of it. You did have a couple of Margarita cocktails at the end of the week.

To answer that question, you only want to focus on a selection of the `total_vector`. In other words, our goal is to select specific elements of the vector. To select elements of a vector (and later matrices, data frames, etc.), you can use square brackets.

Between the square brackets `[ ]`, you indicate what elements to select. For example, to select the first element of the vector, you type `vector[1]`. To select the second element of the vector, you type `vector[2]`.

Notice that the first element in a vector has index 1, not 0 as in many other programming languages.

To select multiple elements from a vector, you can indicate between the brackets what elements should be selected. For example: suppose you want to select the first and the fifth day of the week: use the vector `c(1, 5)` between the square brackets.

Selecting multiple elements of poker_vector with `c(2, 3, 4)` is not very convenient. Many statisticians are lazy people by nature, so they created an easier way to do this: `c(2, 3, 4)` can be abbreviated to `c(2:4)`, which generates a vector with all natural numbers from 2 up to 4.

Another way to tackle the previous exercise is by using the names of the vector elements (Monday, Tuesday, . . .) instead of their numeric positions. For example,

```
poker_vector["Monday"]
```

will select the first element of `poker_vector` since `"Monday"` is the name of that first element.

Just like you did in the previous exercise with numerics, you can also use the element names to select multiple elements, for example:

```
poker_vector[c("Monday","Tuesday")]
```

```
# Assign the poker results of Wednesday to the variable poker_wednesday.
poker_wednesday <- poker_vector[ c("Wednesday") ]
poker_wednesday
```

```
## Wednesday
##        20
```

```
# Select the first and fifth element of poker_vector
poker_vector[ c(1,5) ]
```

```
## Monday Friday
##    140    240
```

```
# Assign the poker results of Tuesday, Wednesday and Thursday to the variable poker_midweek.
poker_midweek <- poker_vector[ c("Tuesday" , "Wednesday" , "Thursday") ]
poker_midweek
```

```
##   Tuesday Wednesday  Thursday
##       -50        20      -120
```

```
# Assign to roulette_selection_vector the roulette results from Tuesday up to Friday; make use of colon
roulette_selection_vector <- poker_vector[ c(2:4) ]

# Select the first three elements in poker_vector by using their names: "Monday", "Tuesday" and "Wednes
poker_start <- poker_vector[ c("Monday" , "Tuesday" , "Wednesday") ]

# Calculate the average of the values in poker_start with the mean() function. Simply print out the res
average <- mean(poker_start)
average
```

```
## [1] 36.66667
```

==== Selection by comparison ====

By making use of comparison operators, we can approach the previous question in a more proactive way.

The (logical) comparison operators known to R are:

- < for less than
- > for greater than
- <= for less than or equal to
- >= for greater than or equal to
- == for equal to each other
- != not equal to each other

Stating `6 > 5` returns `TRUE`. The nice thing about R is that you can use these comparison operators also on vectors.

Working with comparisons will make your data analytical life easier. Instead of selecting a subset of days to investigate yourself (like before), you can simply ask R to return only those days where you realized a positive return for poker.

Now, you would like to know not only the days on which you won, but also how much you won on those days.You can select the desired elements, by putting `selection_vector` between the square brackets that follow `poker_vector`:

`poker_vector[selection_vector]`

R knows what to do when you pass a logical vector in square brackets: it will only select the elements that correspond to `TRUE` in `selection_vector`.

```
# Check which elements in poker_vector are positive (i.e. > 0) and assign this to selection_vector.
selection_vector <- poker_vector > 0

# Print out selection_vector so you can inspect it. The printout tells you whether you won (TRUE) or lo
selection_vector
```

```
##    Monday   Tuesday Wednesday  Thursday    Friday
##      TRUE     FALSE      TRUE     FALSE      TRUE
```

```
# Use selection_vector in square brackets to assign the amounts that you won on the profitable days to
poker_winning_days <- poker_vector[ selection_vector ]
poker_winning_days
```

```
##    Monday Wednesday    Friday
##       140        20       240
```

```
# Create the variable selection_vector2, this time to see if you made profit with roulette for differen
selection_vector2 <- roulette_vector > 0

# Assign the amounts that you made on the days that you ended positively for roulette to the variable r
roulette_winning_days <- roulette_vector[ selection_vector2 ]
roulette_winning_days
```

```
## Wednesday    Friday
##       100        10
```

==== Matrix ====

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the `matrix()` function. Consider the following example:

`matrix(1:9, byrow = TRUE, nrow = 3)`

In the `matrix()` function:

- The first argument is the collection of elements that R will arrange into the rows and columns of the matrix. Here, we use 1:9 which is a shortcut for `c(1, 2, 3, 4, 5, 6, 7, 8, 9)`.
- The argument `byrow` indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we just place `byrow = FALSE`.
- The third argument `nrow` indicates that the matrix should have three rows.

TASK 1: Construct a matrix It is now time to get your hands dirty. In the following exercises you will analyze the box office numbers of the Star Wars franchise. May the force be with you!

In the editor, three vectors are defined. Each one represents the box office numbers from the first three Star Wars movies. The first element of each vector indicates the US box office revenue, the second element refers to the Non-US box office (source: Wikipedia).

TASK 2: Naming a matrix To help you remember what is stored in `star_wars_matrix`, you would like to add the names of the movies for the rows. Not only does this help you to read the data, but it is also useful to select certain elements from the matrix.

Similar to vectors, you can add names for the rows and the columns of a matrix

`rownames(my_matrix) <- row_names_vector`

`colnames(my_matrix) <- col_names_vector`

We went ahead and prepared two vectors for you: `region`, and `titles`. You will need these vectors to name the columns and rows of `star_wars_matrix`, respectively.

TASK 3: Calculating the worldwide box office The single most important thing for a movie in order to become an instant legend in Tinseltown is its worldwide box office figures.

To calculate the total box office revenue for the three Star Wars movies, you have to take the sum of the US revenue column and the non-US revenue column.

In R, the function `rowSums()` conveniently calculates the totals for each row of a matrix.

TASK 4: Adding a column for the Worldwide box office You can add a column or multiple columns to a matrix with the `cbind()` function, which merges matrices and/or vectors together by column.

Just like every action has a reaction, every `cbind()` has an `rbind()`. Moreover, just like `cbind()` has `rbind()`, `colSums()` has `rowSums()`. Let's now calculate the total box office revenue for the entire saga.

```r
# Construct a matrix with 3 rows containing the numbers 1 up to 9, filled column-wise.

# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.9)
return_jedi <- c(309.306, 165.8)
```

```r
# Create a vector box_office to the three vectors into one vector. Call this vector box_office.
box_office <- c( new_hope , empire_strikes , return_jedi )
box_office  # vector 1x6 elements
```

```
## [1] 460.998 314.400 290.475 247.900 309.306 165.800
```

```r
# Construct star_wars_matrix with 3 rows, where each row represents a movie. Name the resulting matrix
star_wars_matrix <- matrix(box_office , nrow = 3 , byrow = TRUE)
star_wars_matrix
```

```
##         [,1]  [,2]
## [1,] 460.998 314.4
## [2,] 290.475 247.9
## [3,] 309.306 165.8
```

```r
# Vectors region and titles, used for naming
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")

# Name the columns with region
colnames(star_wars_matrix) <- region # put rownames colnames with vector

# Name the rows with titles
rownames(star_wars_matrix) <- titles
# Print out star_wars_matrix
star_wars_matrix
```

```
##                              US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

```r
# Calculate the worldwide box office figures for the three movies and put these in the vector named wor
worldwide_vector <- rowSums(star_wars_matrix) # row sums
worldwide_vector
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                 775.398                 538.375                 475.106
```

```r
# Bind the new variable worldwide_vector as a column to star_wars_matrix
star_wars_matrix <- cbind(star_wars_matrix , Worldwide = worldwide_vector) # bind old matrix with new r

# Calculate the total revenue for the US and the non-US region and assign total_revenue_vector. You can
total_revenue_vector <- colSums(star_wars_matrix) # bind new column later

# Bind the new variable total_revenue_vector as a row to all_wars_matrix
all_wars_matrix <- rbind(star_wars_matrix , total_revenue_vector)
all_wars_matrix
```

```
##                               US non-US Worldwide
## A New Hope               460.998  314.4   775.398
## The Empire Strikes Back  290.475  247.9   538.375
## Return of the Jedi       309.306  165.8   475.106
## total_revenue_vector    1060.779  728.1  1788.879
```

==== Selection of matrix elements ====

Similar to vectors, you can use the square brackets `[ ]` to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate the rows you want to select from the columns. For example:

- `my_matrix[1,2]` selects the element at the first row and second column.
- `my_matrix[1:3,2:4]` results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.

If you want to select all elements of a row or a column, no number is needed before or after the comma, respectively:

- `my_matrix[,1]` selects all elements of the first column.
- `my_matrix[1,]` selects all elements of the first row.

==== Arithmetic with matrices ====

Similar to what you have learned with vectors, the standard operators like `+`, `-`, `*`, `/`, etc. work in an element-wise way on matrices in R.

Just like `2 * my_matrix` multiplied every element of `my_matrix` by two, `my_matrix1 * my_matrix2` creates a matrix where each element is the product of the corresponding elements in my_matrix1 and my_matrix2. (Element-Wise Multiplication)

Those who are familiar with matrices should note that this is not the standard matrix multiplication for which you should use `%*%` in R.

```
# Select the non-US revenue for all movies (the entire second column of all_wars_matrix), store the res
non_us_all <- all_wars_matrix[ , 2] # all row , only second column
non_us_all
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                   314.4                   247.9                   165.8
##    total_revenue_vector
##                   728.1
```

```
# Use mean() on non_us_all to calculate the average non-US revenue for all movies. Simply print out the
mean_non_us <- mean(non_us_all)
mean_non_us
```

```
## [1] 364.05
```

```
# This time, select the non-US revenue for the first two movies in all_wars_matrix. Store the result as
non_us_some <- all_wars_matrix[1:2 , 2]  #get row 1-2 , column 2 (non-us)
non_us_some
```

```
##              A New Hope The Empire Strikes Back
##                   314.4                   247.9
```

```r
# Use mean() again to print out the average of the values in non_us_some
mean_non_us_some <- mean(non_us_some)
mean_non_us_some
```

```
## [1] 281.15
```

End-of-File
Pongsun B.
2023-03-01