huber-intro-1

Welcome to this swirl course. If you find any errors or if you have suggestions for improvement, please let me know via stephan.huber@hs-fresenius.de .

The RStudio interface consists of several windows. You can change the size of the windows by dragging the grey bars between the windows. We'll go through the most important windows now.

Bottom left is the Console window (also called command window/line). Here you can type commands after the > prompt and R will then execute your command. This is the most important window, because this is where R actually does stuff.

Top left is the Editor window (also called script window). Here collections of commands (scripts) can be edited and saved. When you do not get this window, you can open it with 'File' > 'New' > 'R script'.

Just typing a command in the editor window is not enough, it has to be send to the Console before R executes the command. If you want to run a line from the script window (or the whole script), you can click 'Run' or press 'CTRL+ENTER' to send it to the command window.

The shortcut to run send the current line to the console is _____.

1. CTRL+SHIFT
2. CTRL+ENTER
3. CTRL+SPACE
4. SHIFT+ENTER

CTRL+ENTER

If you are a Mac user, your shortcut is 'Cmd+Return' instead of 'SHIFT+ENTER'.

Top right is the environment window (a.k.a workspace). Here you can see which data R has in its memory. You can view and edit the values by clicking on them.

Bottom right is the plots / packages / help window. Here you can view plots, install and load packages or use the help function.

The first thing you should do whenever you start Rstudio is to check if you are happy with your working directory. That directory is the folder on your computer in which you are currently working. That means, when you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory.

You can check your working directory with the function getwd(). So let's do that. Type in the command window 'getwd()' .

```
getwd()
```

```
## [1] "/home/sthu/Dropbox/hsf"
```

Are you happy with that place? if not, you should set your working directory to where all your data and script files are (or will be). Within RStudio you can go to 'Session' > 'Set working directory' > 'Choose directory'. Please do this now.

Instead of clicking, you can use the function setwd("/YOURPATH"). For example, setwd("/Users/MYNAME/MYFOLDER") or setwd("C:/Users/jenny/myrstuff"). Make sure that the slashes are forward slashes and that you do not forget the apostrophes. R is case sensitive, so make sure you write capitals where necessary.

Whenever you want R to do something you need to use a function. It is like a command. All functions of R are organized in so-called packages or libraries. With the standard installation many packages are already installed. However, many more exist and some of them are really cool. For example, with installed.packages() all installed packages are listed. Or, with swirl(), you started swirl.

Of course, you can also go to the Packages window at the bottom right. If the box in front of the package name is ticked, the package is loaded (activated) and can be used. To see via Console which packages are loaded type in the console '(.packages())'

```
(.packages())
```

```
## [1] "swirl"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

There are many more packages available on the R website. If you want to install and use a package (for example, the package called 'geometry') you should first install the package. Type install.packages("geometry") in the console. Don't be afraid about the many messages. Depending on your PC and your internet connection this may take some time.

```
install.packages("geometry")
```

```
## Installing package into '/home/sthu/R/x86_64-pc-linux-gnu-library/4.1'
## (as 'lib' is unspecified)
```

After having installed a package, you need to load the package. That is a bit anoying but essential. Type in library(geometry) in the Console. You also did this for the swirl package (otherwise you couldn't have been doing these exercises).

```
library(geometry)
```

Check if the package is loaded typing '(.packages())'

```
(.packages())
```

```
## [1] "geometry"  "swirl"     "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
```

Now, let's get started with the real programming.

R can be used as a calculator. You can just type your equation in the command window after the >. Type 10^2 + 36.

```
10^2 + 36
```

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

And R gave the answer directly. By the way, spaces do not matter.

If you use brackets and forget to add the closing bracket, the > on the command line changes into a +. The + can also mean that R is still busy with some heavy computation. If you want R to quit what it was doing and give back the >, press ESC.

You can also give numbers a name. By doing so, they become so-called variables which can be used later. For example, you can type in the command window A <- 4 .

```
A <- 4
```

The '<-' is the so-called assignment operator. It allows you to assign data to a named object in order to store the data.

Don't be confussed about the term object. All sorts of data are stored in so-called objects in R. All objects of a session are shown in the Environment window. In the second part of this course, I will introduce different data types.

You can see that A appeared in the environment window in the top right corner, which means that R now remembers what A is.

You can also ask R what A is. Just type A in the command window.

```
A
```

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

You can also do calculations with A. Type A * 5 .

```
A*5
```

```
## [1] 20
```

If you specify A again, it will forget what value it had before. You can also assign a new value to A using the old one. Type A <- A + 10 .

```
A <- A + 10
```

You can see that the value in the environment window changed.

To remove all variables from R's memory, type rm(list=ls()) .

```
rm(list=ls())
```

You see that the environment window is now empty. You can also click the broom icon (`clear all`) in the environment window. You can see that RStudio then empties the environment window. If you only want to remove the variable A, you can type rm(A).

Like in many other programs, R organizes numbers in scalars (a single number, 0-dimensional), vectors (a row of numbers, also called arrays, 1-dimensional) and matrices (like a table, 2-dimensional).

The A you defined before was a scalar. To define a vector with the numbers 3, 4 and 5, you need the function c, which is short for concatenate (paste together). Type B=c(3,4,5) .

```
B=c(3,4,5)
```

If you would like to compute the mean of all the elements in the vector B from the example above, you could type (3+4+5)/3. Try this

```
(3+4+5)/3
```

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

But when the vector is very long, this is very boring and time-consuming work. This is why things you do often are automated in so-called functions. For example, type mean(x=B) and guess what this function mean() can do for you.

```
mean(x=B)
```

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

Within the brackets you specify the arguments. Arguments give extra information to the function. In this case, the argument x says of which set of numbers (vector) the mean should be computed (namely of B). Sometimes, the name of the argument is not necessary; mean(B) works as well. Try it.

```
mean(B)
```

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

Compute the sum of 4, 5, 8 and 11 by first combining them into a vector and then using the function sum. Use the function c inside the function sum.

```
sum(c(4,5,8,11))
```

```
## [1] 28
```

The function rnorm, as another example, is a standard R function which creates random samples from a normal distribution. Type rnorm(10) and you will see 10 random numbers

```
rnorm(10)
```

```
##  [1]  0.76977837 -0.14622141  0.53924840 -0.88086971  1.92077037 -0.28015385
##  [7]  0.01594662  1.64208469 -0.66669651  1.08656820
```

Here rnorm is the function and the 10 is an argument specifying how many random numbers you want - in this case 10 numbers (typing n=10 instead of just 10 would also work). The result is 10 random numbers organised in a vector with length 10.

If you want 10 random numbers out of normal distribution with mean 1.2 and standard deviation 3.4 you can type rnorm(10, mean=1.2, sd=3.4). Try this.

```
rnorm(10, mean=1.2, sd=3.4)
```

```
##  [1]  3.56916212 -0.50465437  0.95331590  0.07725623 -5.56529257  2.53201285
##  [7]  0.15829096  6.05524550 -3.35290698 -0.03229734
```

This shows that the same function (rnorm) may have different interfaces and that R has so called named arguments (in this case mean and sd).

Comparing this example to the previous one also shows that for the function rnorm only the first argument (the number 10) is compulsory, and that R gives default values to the other so-called optional arguments. Use the help function to see which values are used as default by typing ?rnorm.

```
?rnorm
```

You see the help page for this function in the help window on the right. RStudio has a nice features such as autocompletion and snapshots of the R documentation. For example, when you type rnorm(in the command window and press TAB, RStudio will show the possible arguments.

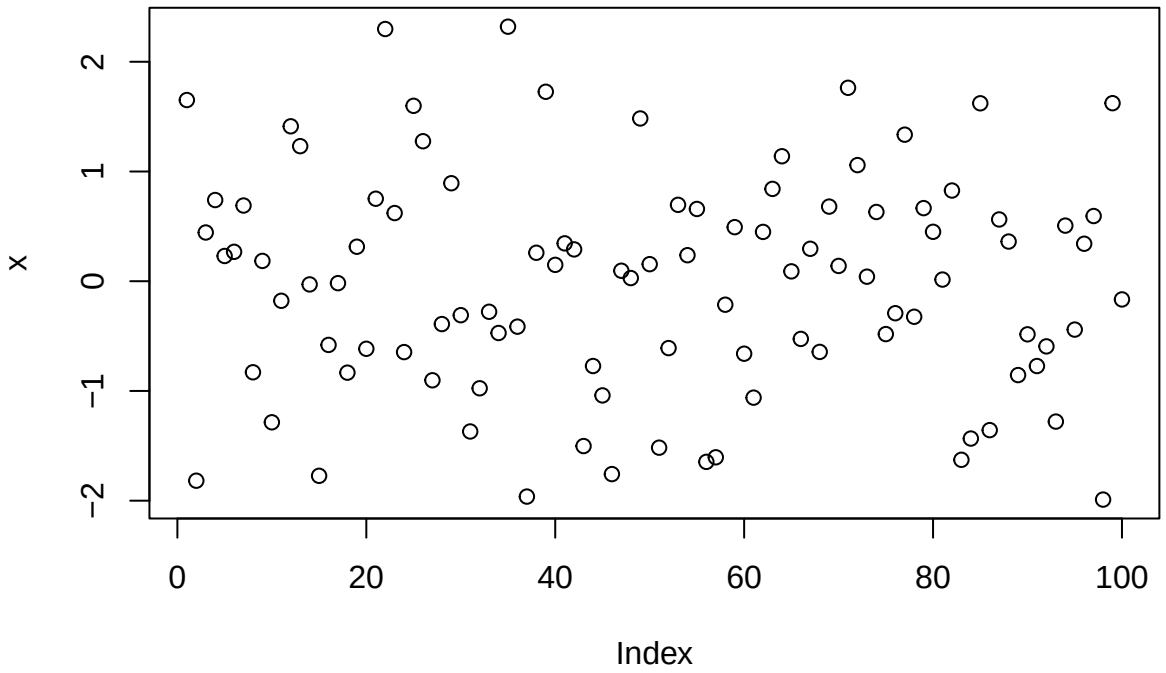
You can also store the output of the function in a variable. Type x=rnorm(100).

```
x=rnorm(100)
```

Now 100 random numbers are assigned to the variable x, which becomes a vector by this operation. You can see it appears in the Environment window.

R can also make graphs. Type plot(x) for a very simple example.

```
plot(x)
```



The 100 random numbers are now plotted in the plots window on the right.

You now are more familiar to RStudio and you know some basic R stuff. In particular, you know...

...that everything in R is said with functions,

...that functions can but don't have to have arguments,

...that you can install packages which contain functions,

...that you must load the installed packages every time you start a session in RStudio, and

...that this is just the beginning. Thus, please continue with the second module of this introduction.