Introduction to R

Introduction to Computational BioStatistics with R

Alexey Fedoseev

September 17, 2024

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Institute of Medical Science
UNIVERSITY OF TORONTO

Computer programs

A computer program consists of *instructions* that tell the computer what to do and *data* that the program uses when it is running.

The data consists of constants or fixed values that never change and variable values. Usually, both constants and variables are defined as certain data types.

Each data type prescribes and limits the form of the data.

Examples of data types include: an integer expressed as a decimal number, or a string of text characters.

Programming languages

You tell computer what to do using a programming languge. There are many programming languages and each of them has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

Regardless of what language you use, eventually your program has to be converted into the machine language so that the computer can understand it. There are two ways to do this:

- Compile the program.
- Interpret the program.

Interpreter versus Compiler

An *interpreter* translates high-level instructions into an intermediate form, which it then executes.

In contrast, a *compiler* translates high-level instructions directly into machine language.

Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long.

The interpreter, on the other hand, can immediately execute high-level programs.

We will begin by using R as the programming language in this class.

R history

R has been around for a while, and is well-developed:

- Introduced in 1996 as an evolution of the S language.
- R is designed for exploring and analysing data.
- Home page: http://www.r-project.org.
- Community packages are stored at CRAN - Comprehensive R Archive Network.
- Bioinformatics packages are also stored at http://bioconductor.org.
- A new full version of R is released each year.

About R

R is used widely in a number of disciplines, like ecology, biology, etc., and provides a solid platform for beginner scientific programmers. Many graphing and statistical operations are built right into the language.

It is a broadly useful language. This means that you should be able to do most anything you want to do using R, and it should be relatively easy to accomplish.

It runs on all major operating systems.

It is used by many scientific programmers and taught in many research science departments. This makes collaborating with experts on large or complex projects much easier.

Starting R

In order to start R open a terminal and type R if you have Linux or an Apple computer, or double-click on the R symbol if you have a Windows on the computer.

Raise your hand if you think it's not working. You are welcome to follow along by entering the commands on the slides, and playing with the output.

There are several graphical R interfaces available. These are handy, but we generally don't recommend them as in some cases they have serious drawbacks. However, you are welcome to use them if you like.

Version of R

After starting R the first thing you see is a version of R you are using.

```
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
```

This is an important information because the developers of R constantly instroducing new features and if you write a program using the latest version of R and give it to someone who is using a very old version of R, it might not work due to the changes made in the language.

R prompt

After reading the greeting message you will find at the very bottom of it a greater sign ‘>’ followed by the cursor. Here you can type in your instructions. R executes them immediately after you hit Enter. Let us try to give our first instruction.

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> "hello"
[1] "hello"
>
```

As soon as the execution of instruction is finished, R displays the result of it and waits for new commands with a new prompt ‘>’.

Data types

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are just reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like integer (1,2,3, . . .) or real numbers (3.1415, 1.2, . . .) , logical values (TRUE, FALSE), etc.. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and Java, in R the variables are not declared as some data type. The variables are assigned with a value and data type of this value becomes the data type of the variable.

Variables in R

Let us create our first variable. First you need to come up with a name of the variable that will describe what you are going to store in this variable.

If I want to store a number in a variable I will call it: `mynumber`. And the value I want to store is: 8. In order to "put" 8 in `mynumber` we use the *assignment* operator: `<-`.

After you finished typing the instruction, press `Enter`.

```
> mynumber <- 8
```

You read "`mynumber <- 8`" as "`mynumber` is assigned the value 8".

Variables in R

Type the name of the variable and press Enter to see the value that is assigned to the variable.

```
> mynumber
[1] 8
```

The value of a variable could consist of several elements and the [1] indicates that the display starts at the first element of variable. We will have more examples later.

Note: The name of a variable in R must start with a letter (A-Z and a-z) and can include letters, digits (0–9), dots (.), and underscores (_). R discriminates between uppercase and lowercase letters in the name of the variable, so that x and X can name two distinct variables.

Data types

As we already know, the data type of the value determines the data type of the variable. So what is the data type of our variable `mynumber`? It is very easy to check.

```
> str(mynumber)
 num 8
```

We can see that R assigned a *numeric* type to our variable. It also showed us the value of this variable.

The command `str` displays the internal **str**ucture of an object in R

Numeric data type

We can check the data type of a number.

```
> str(2)
 num 2
> str(3.14)
 num 3.14
```

As you can see, numeric data type covers both integer and real numbers.

You can safely perform arithmetic operations within numeric data type.

```
> mynumber + 2
[1] 10
> mynumber - mynumber
[1] 0
```

Character data type

A character object is used to represent string values in R. A string is a collection of characters. It means we can put any message in a variable. All you have to do is to put it in quotes.

```
> mygreeting <- "Good Afternoon!"
> mymessage <- "It was a pleasure speaking with you today."
> mysignature <- "Sincerely, John."
> str(mygreeting)
 chr "Good Afternoon!"
```

A very common operation on strings is the operation of joining two strings together called *concatenation*.

Strings concatenation

Let us say you are writing an email. You have a variable `mygreeting` with a greeting phrase, a variable `mymessage` containing the body of your email and a variable `mysignature` holding you signature.

You can use the `cat` command to concatenate all your strings together in one message.

```
> cat(mygreeting, mymessage, mysignature, "\n")
Good Afternoon! It was a pleasure speaking with you today. Sincerely, John.
```

Symbol `\n` is a special character that signifies a **new line**. To fully understand its meaning try running this command without `\n` and see what happens.

Dynamic typing

You can re-use the same variable if you need to adjust the value. Let us say you want to change the greeting in our email.

```
> mygreeting <- "Good Morning!"
>
> cat(mygreeting, mymessage, mysignature, "\n")
Good Morning! It was a pleasure speaking with you today. Sincerely, John.
```

Notice that R allows you to re-assign a different data type to a variable that already has a value. R will overwrite the old value and it will be lost.

```
> str(mynumber)
 num 8
> mynumber <- "416-123-3456"
> str(mynumber)
 chr "416-123-3456"
```

Logical data type

Very often in programming we compare the values. For example:

```
> temperature <- 25
> temperature > 0
[1] TRUE
> temperature < 0
[1] FALSE
```

`TRUE` and `FALSE` are logical values and can be stored in a variable.

```
> is.cold <- (temperature < 0)
> str(is.cold)
 logi FALSE
```

Relational operators

Relational operators allow us to compare values.

```
> temperature
[1] 25
> temperature == 25
[1] TRUE
> temperature > 25
[1] FALSE
> temperature >= 25
[1] TRUE
> temperature <= 0
[1] FALSE
> temperature != 30
[1] TRUE
```

Logical operators

Often we need to combine the results of multiple comparisons. Logical operators are used to perform operations on logical values `TRUE` and `FALSE`.

Logical AND connects two “statements” that are eiter truthful or false. For example: if “it is sunny” AND “it is warm” then “I will go for a walk”. What if “it is sunny” today, but it is -35°C? Then the statement “I will go for a walk” is false.

```
> temperature
[1] 25
> (temperature > 10) & (temperature < 28)
[1] TRUE
> (temperature > 10) & (temperature < 24)
[1] FALSE
> (temperature > -10) & (temperature < 0)
[1] FALSE
```

```
> TRUE & TRUE
[1] TRUE
> TRUE & FALSE
[1] FALSE
> FALSE & TRUE
[1] FALSE
> FALSE & FALSE
[1] FALSE
```

Logical operators

Logical OR helps you decide whether the combination of two “statements” is true or false.

```
> TRUE | TRUE
[1] TRUE
> TRUE | FALSE
[1] TRUE
> FALSE | FALSE
[1] FALSE
```

Logical NOT negates the logical value.

```
> is.cold
[1] FALSE
> !is.cold
[1] TRUE
```

Scripts

Interacting with R using the prompt is a convienient way to play with variables and data. However after you determined your instructions you should write them down separately from the R prompt. Why? Because after you close the prompt you can lose all your progress.

The set of instructions for the interpreted programming language (called the *source code*) saved in a plain text file is called the *script*.

To save the source code always use the plain text editor like Sublime Text, Atom, emacs, vi, nano, etc.. Do not use a heavy text processor like Microsoft Word! It adds a lot of characters to the file that the programming language does not understand.

Let us create our first script in R.

Scripts

Our script will be creating an email. Open the new tab in the text editor and type in the following code

```r
mygreeting <- "Good Afternoon!"
mymessage <- "It was a pleasure speaking with you today."
mysignature <- "Sincerely, John."

cat(mygreeting, mymessage, mysignature, "\n")
```

Save it as `email.R`. The extension `.R` indicates that is it an R script.

Scripts

Now you can run your script by runing the following command in your command line (`bash` for Linux and Apple computers, `git-bash` for Windows)

```
user@scinet scripts $ ls
email.R

user@scinet scripts $ Rscript email.R
Good Afternoon! It was a pleasure speaking with you today. Sincerely, John.
```

As you can see after running the script we are back in the terminal.

Comments

Automating your workflows with a programming language can be very fruitful. You realise it very fast and expand the functionality of your program. Rapidly it can contain hundreds and even thousands of line of code.

However, the common situation is that people tend to simply forget what they meant by a certain notation or calculation in the very beginning.

Using well named variables helps, but sometimes you just want to finish fast and did not give very descriptive names to your variables or put many commands together.

The best practice together with well named variables is to put comments in your code. It means that you put a small description or even your thoughts on what does this line do.

You can put a comment in your program using the # symbol. Everything after that symbol in the line will be treated as a comment and ignored. Comments are for people and not for computers!

```
> # This is a comment and would be ignored by the computer
> mynumber # This is a comment as well
[1] "416-123-3456"
```

Comments

Let us say that you have collected a phone number that consists of 3 variables and you want to display in a specific format.

```r
area.code <- "416" # 416 corresponds to an area code of Toronto
first.digits <- "123"
last.digits <- "3456"

# Merging the parts of a phone number together using "-" as a separator
# Example: 416-123-3456
cat(area.code, "-", first.digits, "-", last.digits, "\n", sep="")
```

Commenting your code is a very good practice. What seems obvious today could be obscure even a week later.

Ask yourself: will I remember what I have done here a year later?