Introduction to programming in R: introduction to R

Erik Spence

SciNet HPC Consortium

8 February 2023

Today's slides

To find today's slides, go to the "Introduction to Programming in R" page, on the right, under Lectures, "Introduction to R".

`https://scinet.courses/1277`

Who am I?

I am Erik Spence.

- I am an Applications Analyst at SciNet (`https://www.scinethpc.ca`).
- SciNet is a High-Performance-Computing (HPC) consortium, one of six in Canada, run by the University of Toronto.
- These consortia run massively parallel computers, with tens of thousands of cores, to perform computations that couldn't be done otherwise.
- My job at SciNet is to help users get their code to run on these machines.
- I also educate users on how to write fast, efficient code.

About this class

Some notes about this class:

- This class aims to be an introductory course on programming using the R language.
- We will meet for twelve weeks, one lecture per week, on Wednesdays, 9:00 - 10:00am.
- Classes will be held in the SciNet teaching room, 661 University Ave., Suite 1140.
- There will be 6 biweekly homework assignments, due one week later at midnight. These will be 100% of your grade.
- Late assignments will be accepted until one week after the deadline (at 9:00am), with a penalty of 0.5 points per day (out of 10).
- Office hours will be held on Wednesdays from 10:00am - 12:00pm (after lecture).
- Please, please ask for help if you need it!
  - Post questions to the class forum.
  - Talk to me, or email me if you have questions: courses@scinet.utoronto.ca.

SciNet certificates

In addition to official UofT class credit, SciNet also offers its own certificates.

- We offer certificates in High Performance Computing, Scientific Computing and Data Science.
- Each certificate requires 36 SciNet credits; specific classes qualify for specific certificates.
- This class qualifies for 9 credits toward the Data Science certificate, and 9 credits toward the Scientific Computing certificate.
- Visit the SciNet education website to see what other courses are available.

`https://scinet.courses`

Class expectations

Some details about the class:

- Prerequisites: minimal-to-no programming experience is sufficient. The goal is to start slowly so all will be on the same page.
- Software you will need:
  - R, and various R libraries.
  - A text editor: Brackets, Sublime, VSCode.
- Grading scheme: the final grade will be based on the homework assignments.
- Attendance is not mandatory, though encouraged. If you don't attend, listen to the recordings! The slides do NOT constitute all of the class material!

Class website

SciNet hosts its own web site for its classes.

- `https://scinet.courses`, or `https://education.scinet.utoronto.ca`
- We will not be using Quercus.
- Log in with your SciNet account, or temporary account.
- Click on "Introduction to Programming in R".
- Or go directly to the class web site: `https://scinet.courses/1277`
- All assignments will be submitted through this website.
- Let me know if you do not yet have an account.

All work for the class will go through this web site, so it is important that you have access.

Student Code of Conduct

Some details about doing the assignments:

- You are welcome to discuss your assignments with each other.
- You are not welcome to copy each other's code.
- You are not welcome to copy code you find on the internet, without giving credit.

`http://tinyurl.com/UofTCodeOfConduct`

Class topics

Our adventure in data analysis will cover the following:

- Getting started with R.
- Vectors, arrays, data frames.
- Modular programming.
- File input/output.
- Visualization.
- Sequences and alignment.
- Other topics.

This list is subject to change. If there's a particular topic that you'd like covered, let me know.

Today's class

Today we will visit the following topics:

- Introduction to R.
- Getting R started.
- Primitive data types.
- Lists.
- Vectors and data frames.

The point of today's class is to introduce you to R, and review the major data types. Please stop me if you have a question.

Computer programs

Let's start at the very very beginning: what is a computer program?

- A computer program is a set of instructions which tell the computer what to do.
- Generally speaking, for scientific computing, you define variables, which contain your data, and perform operations on those variables to do the calculations which you need.
- You also define your own personal functions, which you will then re-use over and over with different parameters (different arguments).
- There are about a bazillion programming languages out there, each with their own strengths and weaknesses.

As you know, we will begin by using R as the programming language in this class.

About R

Some important things to know about R:

- R is a scripting language (like the Linux shell), meaning an interpreter executes commands one line at a time (not a compiled language).
- R can be used interactively, with or without an IDE (RStudio).
- R can also be used non-interactively, run through scripts.
- R has a large repository of community packages.
- R is all about data analysis: it is not a general purpose language.
  - Several important features (numerics, visualization) are baked into the language, not add-ons.
  - Not as useful outside of number crunching.
- R is designed with interactive data exploration in mind.
  - Lots of surprising things "just work" interactively.
  - But this design can make it a little difficult to debug large non-interactive programs.

Starting R

Start R now. This means opening a terminal and typing "R" (Macs), or double-clicking on the R symbol (Windows).

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

Alternatively, 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.

R data types: numeric

Once you start your session you will get an interactive prompt:

- This is the "R" prompt.
- As you enter commands the interpreter interprets them.
- The "<-" symbol is the assignment operator, thus creating a variable.
- As we saw with the Linux shell, variables hold values that we want to use later.
- You should read "a <- 1" as "a is assigned the value 1".
- If you just type the name of the variable, and hit Enter, the value of the variable will be printed.
- The "[1]" is the index of the answer.

```
> a <- 1
> b <- 1.73
>
> a
[1] 1
>
> b
[1] 1.73
>
> a + b
[1] 2.73
>
> d <- a - b
>
> d
[1] -0.73
>
```

Assigning values, an aside

As said on the previous slide:

- You should read "a <- 1" as "a is assigned the value 1".
- Whatever is on the right-hand side of the "<-" is evaluated first.
- If there is a variable on the right-hand side of the "<-", the value of the variable is put into the calculation, before the value is assigned to the left-hand side.
- In this case we are over-writing the previous value of the 'a' variable with a new value.

If you don't understand what's going on on this slide, please raise your hand.

```
>
> a <- 1
>
> a
[1] 1
>
> a <- a + 1
>
> a
[1] 2
>
> a <- (a / 2) + 3
>
> a
[1] 4
>
```

R has functions

Like all languages, R has built-in functions:

- The "typeof" function "returns" the 'type' of the "argument".
- The "argument" of the function is the thing in the brackets.
- The "return value" of the function is the value which the function "gives back" when it's finished running.
- If a returned value is not assigned to a variable R will just print it.
- If the function takes multiple arguments, they are separated by commas.

```
>
> a
[1] 4
>
> typeof(a)
[1] "double"
>
> b <- typeof(a)
>
> b
[1] "double"
>
> is.numeric(a)
[1] TRUE
>
```

Function return values

Function return values are either assigned to a variable, or printed!

- If a returned value is not assigned to a variable R will just print it.
- This is true whether you are running functions on the R command line, or in a script.
- If your scripts start randomly printing things out, it's likely that you're calling functions which return a value, and the value is not getting assigned to a variable.
- Note that functions are not required to return something.

```
>
> a
[1] 4
>
> typeof(a)
[1] "double"
>
> b <- typeof(a)
>
> b
[1] "double"
>
> is.numeric(a)
[1] TRUE
>
```

R data types: strings

R has the usual non-numeric types as well:

- Values in quotes are called "strings" (collections of characters).
- R accepts single or double quotes.
- "paste" converts the inputs to strings (if it's not already), concatenates them, and returns them.
- "cat" prints the arguments to the screen. It also needs a newline character (\n).
- Notice that these are built-in functions.

```
> e <- "hello"
> g <- 'world'
>
> mystring <- paste(e, g)
>
> print(mystring)
[1] "hello world"
>
> cat(e, g, '\n')
hello world
>
> typeof(e)
[1] "character"
>
> e + g
Error in e + g :  non-numeric argument to
binary operator
```

Strings versus variable names

Don't get confused between variables and strings:

- If a variable is assigned the value of a string, it is assigned the value of a string. The string has nothing to do with any similarly-named variables.
- When a variable is assigned another variable, it's assigned the variable's value.
- If it doesn't have quotes around it, it's either a variable or a function.

```
>
> a <- 1
>
> b <- "a"
>
> b
[1] "a"
>
> d <- a
>
> d
[1] 1
>
```

R data types: booleans

R has the usual non-numeric types as well:

- TRUE and FALSE are called "boolean" values (sometimes called "logical").
- The "!" is the NOT operator. It returns the opposite of the argument.
- If you're feeling clever, you can do math on booleans.

```
> f <- FALSE
>
> f
[1] FALSE
>
> !f
[1] TRUE
>
> typeof(f)
[1] "logical"
>
> is.logical(f)
[1] TRUE
>
> TRUE + TRUE
[1] 2
>
```

R lists

Lists are the most basic "container" data type in R:

- The "list" function will generate a list from the inputs.
- Lists can be of mixed type.
- "pi" is a builtin variable.
- "str" stands for "structure". It gives a description of the argument.

```
>
> l <- list(a, b, e, f, g, pi)
>
> str(l)
List of 6
$ : logi TRUE
$ : chr "a"
$ : chr "hello"
$ : logi FALSE
$ : chr "world"
$ : num 3.14
>
> is.list(l)
[1] TRUE
>
```

R lists, continued

The values of lists can be of various types, including other lists.

- Accessing individual items in a list is done with [[ ]].
- The number in the double square brackets is called the "index".
- Indexing starts at 1, as with most scientific computing languages.
- In R, the "start:finish" notation returns a sequence running from start to finish, inclusive.

```
>
> l[[6]]
[1] 3.141593
>
> l[1:3]
[[1]]
[1] TRUE

[[2]]
[1] "a"

[[3]]
[1] "hello"
>
```

R named lists

- Named lists allow you to access elements by name, rather than by index.
- If you don't finish your line in R, but hit enter, it will display the "+" symbol, indicating that it's waiting for more input.
- You can access pieces of a named list with the "$".
- The "names" function returns the names of a named list, data frame, etc.

```
>
> named.list <- list(value = 5,
+ word = "text", number = 7.3)
> str(named.list)
List of 3
 $ value : num 5
 $ word  : chr "text"
 $ number: num 7.3
>
> named.list$value
[1] 5
> named.list[["number"]]
[1] 7.3
> names(named.list)
[1] "value"   "word"    "number"
>
```

R vectors

Vectors are baked right into R:

- Homogeneous (same type).
- Compact.
- Not nested.
- The "c" command combines values in to a vector or list.

```
> a <- c(1,2,3)
>
> b <- c("Hello", "World", "From", "A", "Vector")
>
> str(b)
chr [1:5] "Hello" "World" "From" "A" "Vector"
>
```

We'll discuss vectors in more detail next class.

Built-in data sets

R contains built-in data sets that can be used for practicing.

```
> data()
Data sets in package ‘datasets’:

 AirPassengers           Monthly Airline Passenger Numbers 1949-1960
 BJsales                 Sales Data with Leading Indicator
 BJsales.lead (BJsales)  Sales Data with Leading Indicator
 BOD                     Biochemical Oxygen Demand
 ⋮
```

```
>
```

```
> str(faithful)
data.frame’:   272 obs.  of 2 variables:
$ eruptions: num 3.6 1.8 3.33 2.28 4.53 ...
$ waiting  : num 79 54 74 62 85 55 88 85 51 85 ...
```

```
>
```

Type 'q' to get out of the 'data' menu.

Data frames

Data frames are a building block for data analysis in R.

A data frame is a named list of vectors (similar to a spreadsheet). Each vector (a column of the frame) has the same length, but different columns may have different types.

```
>
> my.data <- trees
> class(my.data)
[1] "data.frame"
>
> str(my.data)
'data.frame':  31 obs.  of 3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
>
```

Slicing data frames

Accessing parts of the data frame makes a lot more sense when you remember it's just a named list of vectors.

```
>
> names(my.data)
[1] "Girth" "Height" "Volume"
>
> my.data[1:3,"Girth"]
[1] 8.3 8.6 8.8
>
> my.data[c(2,3,5),]
   Girth  Height  Volume
 2   8.6      65    10.3
 3   8.8      63    10.2
 5  10.7      81    18.8
>
```

Summary

A review of what we covered today.

- R is a programming language dedicated to data analysis.
- R can be run either as a script or interactively.
- R has several primitive data types: numeric, string, boolean.
- Variables can be created which hold your information.
- R has many builtin functions.
- R has container data types: lists, vectors, data frames, and many others.

We'll continue learning how to use R in the next class.