ALGONQUIN COLLEGE

SCHOOL OF ADVANCED TECHNOLOGY

ICT - Applications & Programming
Computer Engineering Technology – Computing Science

Numerical Computing – CST8233

Lab #1 – Installing and Using the R IDE

This course is based around the R language. In order to be successful in this course, you will need to have the tools to write and execute R code.

Objectives

- Install the R runtime,
- Install the R IDE, RStudio,
- Demonstrate that everything works by running some simple R code.
- Get familiar with R classes and data types

You will need to show your lab professor to get your grades.

Grades:

1% of your final course mark

Deadline

During the lab period of Week 2 (May 17)

PART I

Step 1. Installing R

Go to https://cran.r-project.org/mirrors.html and choose a download mirror. There are a few in Canada, Simon Fraser University or Waterloo may be good choices.

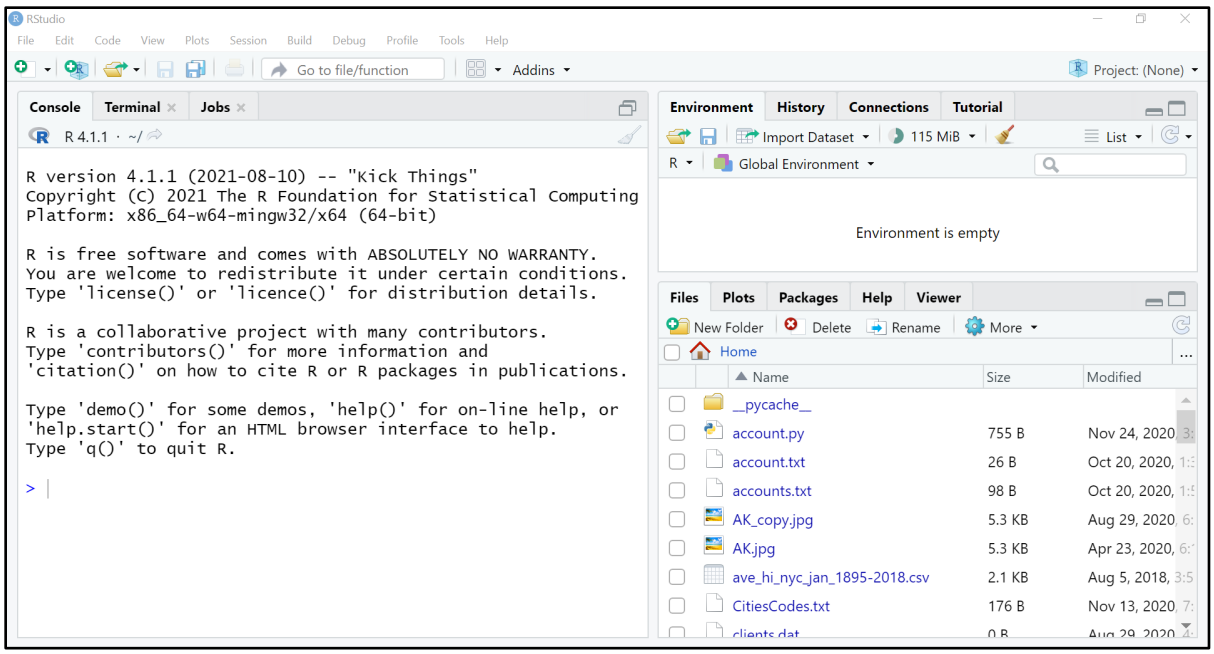
Choose the option based on your operating system. On the page that loads, choose the correct version for your computer. Then choose the *"install R for the first time"* option. The version as of January 6th, 2023 is R-4.2.2 for Windows.

Once it is downloaded, run the installer as *Adminstrator*. Follow the instructions and accept default options. This may take a few minutes.

Step 2. Installing RStudio

To install the RStudio IDE, go to https://www.rstudio.com/products/rstudio/ and choose the *"RStudio Desktop"* option. The regular "Rstudio Desktop" is the one you want. On the next page, choose the correct version for your computer. The current release for Windows 10/11 as of January 6th, 2023 is 2022.12.0–353.

Once you have downloaded it, you may need to run it as *Administrator*. Follow the instructions shown during the setup. Once it is done installing, launch the program. You should see something like this:



Step 3. Get Familiar with RStudio

In order to get familiar with RStudio, watch the following videos:

- RStudio for the Total Beginner. (https://www.youtube.com/watch?v=FIrsOBy5k58)
- Getting started with RStudio. (https://www.youtube.com/watch?v=DuQSQQa6Ssw)

Step 4. Two Examples

In these examples, all lines starting with # are comments. You should be able to run the following code samples.

Example 1. Hello World Program

```
> # We can use the print() function
> print("Hello World!")
[1] "Hello World!"
> # Quotes can be suppressed in the output
> print("Hello World!", quote = FALSE)
[1] Hello World!
> # If there are more than 1 item, we can concatenate using paste()
> print(paste("How","are","you?"))
[1] "How are you?"
```

Example 2. Take input from user

```
my.name <- readline(prompt="Enter name: ")
my.age <- readline(prompt="Enter age: ")
# convert character into integer
my.age <- as.integer(my.age)
print(paste("Hi,", my.name, "next year you will be", my.age+1, "years old."))
```

Run this program with your name.

PART II

R Expressions and Data Types

1. Entering Input

Remember from PART I that "<-" is used as the assignment operator.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> msg <- "hello"
```

After a complete expression is entered, it is evaluated and the result is returned.

```
> x <- 5  ## nothing printed
> x       ## auto-printing occurs
[1] 5
> print(x)  ## explicit printing
[1] 5
```

Note: The "[1]" shown in the output indicates that x is a *vector*, and the output starts at 1. The "5" shown is the first (and in this case, only) element displayed.

Try the following expression:

```
> x <- 10:30
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20 21
[13] 22 23 24 25 26 27 28 29 30
```

Questions for you to consider:

- What does the [13] indicate?
- What does the ":" operator do?

2. R Classes

There are six main classes in R language: character, numeric, integer, complex, logical, and raw.

- *Character*: ‘a’, ‘speed’, ‘TRUE’, ’23.5’.
  You might think of these as "strings" in other languages.
  - fac1 <- ‘speed’ OR Fac1 <- “Speed””

- *Numeric*: 12.5, 909.
  This is for floating-point math.
  - y <- 12.5

- *Integer*: 2L, 5L 0L.
  These are explicitly integers, and must be noted with an L (otherwise they'll be treated as Numeric type instead)
  - int1 <- 8L

- *Complex*: 4+2i.
  This is a form of two-dimensional number, often useful in physics and engineering.
  - com1 <- 6-6i

- *Logical*: TRUE, FALSE.
  You may be familiar with these as booleans in other languages.
  - x <- TRUE

- *Raw*: This holds and displays the data as its "raw" bytes. They may be displayed in hexadecimal. The string “language” is displayed as: 6c 61 6e 67 75 61 67 65.
  - var <- charToRaw(“language”)

3. R Objects

R language has many objects. We will focus on *vectors*, *lists*, *matrices*, *arrays*, *factors*, and *data frames* in this lab.

3a. Vectors

Data is commonly stored as a single-element vector. To create a vector with more than one element, the c() function is used.

```
> x <- c(0.5, 0.6)       ## numeric
> x <- c(TRUE, FALSE)    ## logical
> x <- c(T, F)           ## logical
> x <- c("a", "b", "c")  ## character
> x <- 9:29              ## integer
> x <- c(1+0i, 2+4i)     ## complex
```

The vector() function will initialize a new vector with a given length:

```
> x <- vector("numeric", length = 10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
```

It is possible to combine different types of R objects in the same vector:

```
> y <- c(1.7, "a")   ## character
> y <- c(TRUE, 2)    ## numeric
> y <- c("a", TRUE)  ## character
```

Try the above example and check the class of each created vector using the class() function.

These examples show the effect of implicit coercion. In other occasions, objects need to be explicitly coerced from one class to another. For this, “as.*” functions are used. This is identical to the concept of "casting" in other languages.

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

3b. Lists

Lists behave much as vectors do, but unlike vectors, they are meant to manage elements with different data types. You can create a list by using the list() function, which can take any number of arguments:

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE
```

An empty list of specified length can be created using the vector() function:

```
> x <- vector("list", length = 4)
> x
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL
```

3c. Matrices

In R, matrices are two-dimensional vectors. The matrix() function is used to create a matrix. In the below case, we explicitly state the number of rows and columns, with the nrow and ncol attributes:

```
> m <- matrix(nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

It is also possible to create a matrix from a vector, with the required size attributes:

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

- Note that the order of data entry is top to bottom, left to right (column-wise instead of row-wise).
- It's possible to use the c() function instead of the ":" operator, and wind up with different types of matrices.

Matrices can also be created from vectors by using dim() to specify a dimension:

```
> m <- 1:10
> m
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(m) <- c(2, 5)
> m
```

Try the above example to see what happens.

Finally, matrices can be created by columns or by rows through column-binding and row-binding, using the cbind() and rbind() functions respectively:

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

3d. Arrays

Unlike matrices, which are limited to two dimensions, arrays may be any number of dimensions. Like matrices, arrays can be sized with dim():

```
> a <- array(c('red','yellow'), dim = c(3,3,2))
> print(a)
, , 1

     [,1]     [,2]     [,3]
[1,] "red"    "yellow" "red"
[2,] "yellow" "red"    "yellow"
[3,] "red"    "yellow" "red"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "red"    "yellow"
[2,] "red"    "yellow" "red"
[3,] "yellow" "red"    "yellow"
```

3e. Data Frames

Data frames can store data in table form, not unlike a spreadsheet. Each column may be a different data type, but each column must contain the same number of elements.

Data frames, unlike other elements shown in this document, will also provide headers. The names for each header will be drawn from the vector names, but can be overridden with row.names.

A data frame lends itself very well to reading from CSV (comma-separated value) datasets, which can be read using read.table() or read.csv() functions. They can also be explicitly created with the data.frame() function:

```
> BMI <- data.frame(
+     gender = c("Male", "Male","Female"),
+     height = c(152, 171.5, 165),
+     weight = c(81,93, 78),
+     Age = c(42,38,26)
+ )
> print(BMI)
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
>
```

Alternately:

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

4. Arithmetic and Logical Operators

All the arithmetic operators in R are vectorized. Any operation is done on every element of the vector:

```
c(2, 3, 5, 7, 11, 13) - 2           #subtraction
## [1]  0  1  3  5  9 11
-2:2 * -2:2                         #multiplication
## [1] 4 1 0 1 4
identical(2 ^ 3, 2 ** 3)            #we can use ^ or ** for exponentiation
                                    #though ^ is more common
## [1] TRUE
1:10 / 3                            #floating point division
##  [1] 0.3333 0.6667 1.0000 1.3333 1.6667 2.0000 2.3333 2.6667 3.0000 3.3333
1:10 %/% 3                          #integer division
##  [1] 0 0 1 1 1 2 2 2 3 3
1:10 %% 3                           #remainder after division
##  [1] 1 2 0 1 2 0 1 2 0 1
```

There are three vectorized logical operators in R: “!” is used for logical "not," “&” is used for logical "and," “|” is used for logical "or."

```
> x <- 1:10
> x >= 5
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> !x
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> y <- 1:10
> z <- y %% 2
> z == 0
 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> x & y
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> x | y
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Exercises:

1. Write a code using R language to create three vectors v1, v2, and v3 and initialize them with the following integers:
   a. v1: 2, 4, 6,
   b. v2: 7, 9, 11, and
   c. v3: 13, 15, 17.

   Create a matrix using these three vectors where each column represents one of these vectors. Finally, print this matrix.

2. Write a code using R language to create a data frame which contains details of four students and display them as shown below.

```
        Name Gender Age Designation NoCourses
1  Michael A      M  18 CET Student         5
2 Jennifer R      F  19  CP Student         4
3     Sara B      F  20 SSN Student      <NA>
4    James H      M  22  CS Student         3
```

Note: You need to demo this before the end of this lab period. If you are running low on time, it is best to demonstrate what you have achieved.

If you have both programs running, demo this to your lab professor to get your lab marks. You must do this before the end of the lab period.

Have fun with R, and remember:

> **"The only way to learn a new programming language is by writing programs in it."** - *Dennis Ritchie*