1.1 Introduction to R

R Team

Some Notes:

1. Avoid modifying the built-in comments (this could break the code)
2. Run all lines of code sequentially, some lines will return an error if the prerequisite code has not been run. (you will come to understand why this is by the end of the level)
3. Remember that it is not necessary for you to understand some sections of the code that are being used for demonstrative purposes.
4. Only write your code in files ending in the ".R" extension.
5. Try your best to code along with the session, participate in exercises, take notes, and ask questions along the way!

What is R?

```
* Open source software, can freely be downloaded from https://www.r-project.org/
* Software environment for statistical computing and graphics
* Designed for making Data Analysis easier
* Overall capabilities are virtually identical to Python, each language has pros and cons
```

Who uses R?

Academia:

- R is largely used in universities for education and research
- Many books on statistics use R as their programming language

Data Scientists & Data Analysts for Cleaning/pre-processing:

- Cleaning data is an important part of any Data Science project (80%)
  - Data Cleaning, Categorical Encoding, and Data Imputing
  - R has an extensive library of tools for these purposes

Business World:

- Reporting
- Visualization
- Machine Learning

Basics

1.1 Comments:

- Comments are mainly used to document code
- By commenting your code, you will know what you intended to do when you look at your code in future.
- Two types:

  - Hashtag (#): used for single line comments
  - double-quotes (" "): Used for multi-line comments

1.2 Installing and loading R packages

What are Packages?

- Sets of functions/objects that help us code faster and easier.
- These packages are written by experts and are published for anyone to use for free.
- Anyone can contribute to these packages or create their own.

1. Installing using the editor: bottom-right/packages/
2. Installing: Installation happens using the built-in function **install.packages('package_name')**

```
install.packages('tidyverse')
install.packages('ggplot2')
```

Loading packages

- In order to use the code in a package, it first has to be imported
- This is done using the library function.

```
library(tidyverse)
library(ggplot2)
```

`Note`: Library, package, and framework are used in Software Design and for most of the part, they mean the same thing.

3. Updating: Maintainers of packages tend to provide version updates and bug fixes so it is prudent to update your installed packages every once in a while.

```
update.packages('tidyverse)
```

1.3 Arithmetic (ET: 5)

- Arithmetic operators: (+ | - | / | ^)

```
1 + 1
```

```
## [1] 2
```

```
10 / 2
```

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

```
67 - 2
```

```
## [1] 65
```

```
2 ^ 2
```

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

- Scientific notation:
  - 0.0000000000000001 = 1e-16
  - Easier to follow
  - Clear

```
0.0000000000000001 == 1e-16
```

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

1.4 Variable declaration (ET: 5)

- There are two ways to declare variables and they have subtle differences, for the sake of simplicity and cohesion, we will use "<-" annotation
- variable_name <- value (or variable_name = value)
- Things to remember when naming variables:
  1. Shouldn't begin an variable name with a number.
  2. Spaces are not allowed.
  3. Everything in programming is case sensitive so if you ever get an error message that includes the term "not found", be sure to double check that you have entered your variable name verbatim.

```
1my_name <- 'other words' # Starting with number
```

```
## Error: <text>:1:2: unexpected symbol
## 1: 1my_name
##      ^
```

```
my name <- 'some words'  # Containing a space
```

```
## Error: <text>:1:4: unexpected symbol
## 1: my name
##        ^
```

More examples:

```
two <- 2
HelloWorld <- 'HelloWorld'
```

Exercise:

1. Assign value 2 to a
2. Assign value "2" to b

1.5 Data types

What are data types? (ET: 2)

In programming language we use numbers, strings, character, and different constructs to implement our logic. These are referred to as data types. Since, for a specific task, one might require a number or a string.

In R, we can identify the data type of any object, using the **typeof** function.

1. Numeric values: (ET: 2)
   - Numbers such as 1,1.34, 9.2e-10
   - typeof will return double for any kind of numeric value (this is not the case for other programming languages)

```
typeof(1)
```

```
## [1] "double"
```

```
- Can preform arithmetic operations on them
```

```
1 + 10 # Two numerical values can be added
```

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

```
1 - 10 # Two numerical values can be subtracted
```

```
## [1] -9
```

```
10 / 5 # Two numerical values can be divided
```

```
## [1] 2
```

```
10 * 5 # Two numerical values can be multiplied
```

```
## [1] 50
```

Exercise: Identify if each of the following are numerical (ET: 1/3)

- 1

- 1e-12
- "abcd"
- TRUE

2. Logical Values: (ET: 5)

- The result of evaluating an expression
- Used to build logic in the program
- Operators:
  - "==" is used to check for equality
  - ">", "<", "<=", ">=" used for greater/less than (equal) to

```
10 > 2
```

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

```
5 < 6
```

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

```
5 <= 6
```

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

```
10 >= 2
```

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

```
(10-8) == 2
```

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

```
10 %% 2 == 1 # Gets the modular
```

```
## [1] FALSE
```

- `T/TRUE/1`: The evaluated expression is True.
- `F/False/0`: The evaluated expression is False.
- `NOTE`: True, False, t and f are not the logical operators

```
TRUE == True
```

```
## Error in eval(expr, envir, enclos): object 'True' not found
```

```
FALSE == False
```

```
## Error in eval(expr, envir, enclos): object 'False' not found
```

- Combinations:
  - &&:
    - AND operator
    - All the expression separated by && should be True
    - If one of the expressions is FALSE then it will return False
  - ||:
    - OR operator
    - At least one of the expressions should be True

        * If all the expressions are False then it will return False
    - Negation: "!" negates the results

```
!TRUE == FALSE
```

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

Exercise: Evaluate the result of following expressions

- (FALSE || TRUE) && FALSE
- (TRUE && FALSE) && TRUE
- (!FALSE || FALSE)

3. Characters (ET: 5)
    - Represents strings and single letters as well as other symbols
    - Lower/Upper case:
        - tolower(string): changes Upper letters to lower letters
        - toupper(string): opposite of tolower, lower -> upper

```
tolower('ABCD')
```

```
## [1] "abcd"
```

```
toupper('abcd')
```

```
## [1] "ABCD"
```

```
  - nchar(string): counts the number of characters in a string
```

```
nchar("Galaxy")
```

```
## [1] 6
```

```
  -  Concatenation:paste(R, is Awesome!!): concatenates the strings in the given
     order with a space in between
```

```
paste("R", "is Awesome!!")
```

```
## [1] "R is Awesome!!"
```

1.6 Lists

```
a <- c(1,2,3)

a
```

```
## [1] 1 2 3
```

sum the list:

```
sum(a)
```

```
## [1] 6
```

1.7 For Loop

Loops are used to build the logic of the program by implementing constructs within the code.

```
# Prints the numbers one to five.
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Exercise 1. Print the Squared of each number from 1 to 20. 2. Find the biggest value in the list [1,20,50,7,2,132,56,12] using a For loop.

1.8: Functions

Functions have numerous use cases in programming.

- Automating processes
- Help us write cleaner code
- Build logic

The format for making functions is simple:

1. First name your function (add)
2. Then assign a fucntion() to it.
3. Pass in some parameters in the function argument.
4. Implemenet your logic in the curly brackets.

```
# Name of the function "<-" function(parameters...) {what the function does!}
add <- function(x, y) {
  x + y
}

add(1,2)
```

```
## [1] 3
```

Built-in functions:

Functions/Objects that are available in the base R package do not require external installation

- **sqrt(x)**: Takes the square root of values

```
sqrt(4)
```

```
## [1] 2
```

- **log(x, base)**: returns logarithm of a value with a given base

```
log(9,3)
```

```
## [1] 2
```

```
log(216,6)
```

```
## [1] 3
```

Exercise 1. Write a function that gets a number and returns True if it is an odd value and False otherwise. 2. Write a function that computes the factorial value for a given number.

```
ex1 <- function(n) {
  return(n %% 2 == 0)
}

ex2 <- function(n) {
```

```
  num = 1
  for (i in 2:n) {
    num = num * i
  }

  return(num)
}
```