**GENERAL**

- R version 3.0 and greater adds support for 64 bit integers
- R is case sensitive
- R index starts from 1

**HELP**

- `help(functionName)` or `?functionName`
- `help.start()`
- `example(topic)`
- `apropos('mea')`
- `demo(topic)`

**SYMBOL NAME ENVIRONMENT**

- If multiple packages use the same function name, the function that the package loaded last will get called.
- To avoid this precede the function with the name of the package, e.g. `package::functionName`

**LIBRARY**

- Only trust reliable R packages i.e., for plotting, 'sp' for dealing spatial data, 'reshape2', 'survival', etc.
- Library
- To avoid this precede the function with the name of the symbOL NamE ENvirONmENt

**DATA TYPES**

- **Check data type**: `class(variable)`
- **FOUR BASIC DATA TYPES**
  1. **Numeric** - includes float/double, int, etc. `is.numeric(variable)`
  2. **Character(string)**
  3. **Date/POSIXct**
  4. **Logical** - (TRUE = 1, FALSE = 0) `as.logical(EXPR)`

**MANIPULATING STRINGS**

- **Putting Together Strings**
  - `paste('string1', 'string2', sep = '')`
  - `paste(c('1', '2'), collapse = ' ')` returns '1 2'

- **Split String**
  - `str_split(string = v1, pattern = '\-?')` returns a list

- **Get Substring**
  - `substr(string = v1, start = i, end = 3)`

- **Match String**
  - `is.JohnFound <- function(string, pattern = ignore.case('John'))` returns True/False if John was found

**OBJECTS in current environment**

- Display Object Name `objects() or ls()
- Remove Object `rm(object1, object2, ..)

**DATA Structures**

**VECTOR**

- Group of elements of the SAME type
- R is a vectorized language, operations are applied to each element of the vector automatically
- R has no concept of column vectors or row vectors
- Special vectors: letters and LETTERS, that contain lower-case and upper-case letters

**FACTOR**

- `as.factor(vector)` gets you the levels which is the number of unique values
- Factors can reduce the size of a variable because they only store unique values, but could be buggy if not used properly

**LIST**

- Store any number of items of ANY type
- `list1 <- list(first = 'a', second = 1)`
- `list1['newElement'] <- 2`
- `list1[[1]]` gets you the levels which is the first

**Differences: data.table vs. data.frame**

- By default, data.frame turns character data into factors, while data.table does not
- When you print data.frame, data all data prints to the console, with a data.table, it intelligently prints the first and last five rows

**KEY Difference: Data.tables are fast because they have an index like a database.**

- i.e., this search, `df1$col > number`, does a sequential scan (vector scan). After you create a key for this, it will be much faster via binary search.

**MATRIX**

- Similar to data.frame, except every element must be of the SAME type, most commonly all numerics
- Functions that work with data.frame should work with matrix as well

**ARRAY**

- Multidimensional vector of the SAME type
- `array <- array(1:2, dim = c(2,3,2))`
- Using arrays is not recommended
- Matrices are restricted to two dimensions while array can have any dimension
**DATA MUNGING**

**APPLY** (apply, apply, apply, apply)
- Apply - most restrictive. Must be used on a matrix, all elements must be the same type
- If used on some other object, such as a data.frame, it will be converted to a matrix first.
- **apply(matrix, 1 - rows or 2 - columns, function to apply)**
  - If rows, then pass each row as input to the function
- By default, computation on NA (missing data) always returns NA, so if a matrix contains NAs, you can ignore them (use na.rm = TRUE in the apply(..) which doesn’t pass NAs to your function)

**lapply**
Applies a function to each element of a list and returns the results as a list

**sapply**
Same as lapply except return the results as a vector

**HELPERS**
- Each() - supply multiple functions to a function like aggregate
- Do.call() - specify the name of a function either as string or as a symbol

**ARGUMENTS**
- If cell separator has been used inside a cell, then use `Read_csv(file = 'path'` instead of `read.csv()` or `read.delim()` instead of `read.table()`

**LOAD DATA FROM CSV**
- `read.csv()`
- `read.table()` header = TRUE
- `read.csv()` & `read.delim()` return a vector by default

**DATABASE**
- `db <- RODBC:odbcConnect('conStr')`
- `db <- RODBC:sqlQuery(db, 'SELECT ...', stringAsFactors = FALSE)`
- `Close Connection: RODBC::odbcClose(db)`
- `Only one connection may be open at a time. The connection automatically closes if R closes or another connection is opened.`
- `If table name has space, use [ ] to surround the table name in the SQL string. If [ ] in a column, then use the function to replace it with a space (e.g., replace(' ', '_'))
- `which()` in R is similar to ‘where’ in SQL

**INCLUDED DATA**
- R and some packages come with data included.
- `data()`, `data()`

**MISSING DATA**
- `is.na()`, `is.na()`

**FUNCTIONS AND CONTROLS**

**CREATE FUNCTION**
- `say_hello <- function(first, last = 'bola')` (first: first name, last: last name)
- `Call Function` (call the function)

**HELPERS**
- Each() - supply multiple functions to a function like aggregate
- Do.call() - specify the name of a function either as string or as a symbol

**AGGREGATE (SQL GROUPBY)**
- `aggregate(formula, data, function)`
- Formulas: y ~ x, y represents a variable that we want to make a calculation on, x represents one or more variables we want to group the calculation by.
- Can only use one function in aggregate(). To apply more than one function, use the plyr() package

**PLYR (split-apply-combine)**
- `ddply()`, `lply()`, `ldply()`, etc. (1st letter is the type of input, 2nd is the type of output)
- plyr can be slow, most of the functionality in plyr can be accomplished using base function or other packages, but plyr is easier to use

**DPLYR (for data.frame ONLY)**
- Basic functions: filter(), slice(), arrange(), select(), rename(), distinct(), mutate(), summarise()