R Data Structures

R Vectors:

Vector is a basic data object in R and can contain elements of same or different data types.

A vector containing elements of same data type is known as an atomic vector.

Syntax:

```
variable <- c(elements separated by comma)
```

Ex: Atomic vector of type numeric

```
value <- c( 1, 2.1, 3.2 )
```

Atomic vector of type character

```
value <- c( "banana", "apple", "orange")
```

Non atomic vectors

```
value <- c( 1, 3+20i, "cat", "R" )
```

Vector with numerical values in a sequence

```
value <- 1:10

value <- seq (from = 0, to = 100, by = 20)
          seq ( start, stop, step )
```

Repeat each value

output :

```
value <- rep ( c (1,2,3) , each = 3 )
cat (value)
```

```
1 1 1 2 2 2 3 3 3
```

Repeat sequence of vector

```
value <- rep(c(1,2,3) , times = 3)
cat(value)
```

```
1 2 3  1 2 3  1 2 3
```

Repeat each value independently

```
value <- rep(c(1,2,3) , times = c(4, 2, 1)
cat ( value)
```

```
1 1 1 1 2 2 3
```

Accessing Vector Elements :

Indexing starts at 1

You can access the vector elements by referring to its index position inside square brackets

Syntax : variable [index]

output :

Example:

```
value <- c(11, 14, 3, 1 , 9)
cat (value [2])
```

```
14
```

To modify the element in a vector, refer to the index position

Ex:

```
                                              output:
value <- c(11, 14, 3, 1, 9)
value [4] <- 7
cat (value)                                   11  14  3  7  9
```

To access all elements but not the element at index, idx, use negative indexing

Ex:

```
cat (value [-2])                              11  3  7  9
```

displays all the elements except for the element at index 2.

Note: You can add, subtract, multiply, divide vectors, also compare two vectors provided the vectors are of same length.

To find the number of elements in the vector, use the inbuilt length() function.

```
                                              output:
value <- c(11, 14, 3, 1, 9)
cat (length(value))                           5
```

R Listo :

A list is a combination of elements of any data type. A list can contain strings, numbers, vectors, etc, or even other lists

Syntax :

variable <- list (elements separated by comma)

Ex:

```
my-list <- list (40, "Joy.", 2.43, c(1,2,3), "R")

cat (my-list [4])                          output :
                                           1  2  3
```

* To check if a specific element is present in a list, use %in% operator

```
                                           Output:
x <- "apple" %in% my-list                  FALSE
```

* To add an element at the end of a list

```
                                           output
my-list <- list( 1, 2, 3, 4, 5)
append (my-list , 8)
cat (my-list)                              1 2 3 4 5 8
```

* To add an element to the right of a specified index

```
append (my-list , "apple" , after = 2)
cat (my-list)                              1 2 apple 3 4 5 8
```

R Factors :

These are implemented to categorize the data or represent categorical data and store it on multiple levels.

They can be stored as integers with a corresponding label to every unique integer.

The R factors may look similar to character vectors, they are integers and care must be taken while using them as strings .

The R factor accepts only a restricted number of distinct values . For example

- * Music : Rock , pop , classic, jazz
- * Training : Strength , Stamina
- * Gender : Male , Female, Transgender

Attributes of Factors in R :

x : It is the vector that needs to be converted into a factor

levels : It is a set of distinct values which are given to the input vector x

labels : It is a character vector corresponding to the number of levels

exclude : This mentions all the values you want to exclude

ordered : This logical attribute decides whether the levels are ordered

nmax : It will decide the upper limit for the maximum number of levels

Syntax :

```
factor_variable <- factor (vector)
```

Ex:

```
val_1 <- c("FWD", "RWD", "AWD", "FWD", "AWD", "4WD")
fac-val_1 <- factor (val-1)
print (levels(fac-val-1))
```

```
output :   "4WD"  "AWD"  "FWD"  "RWD"
```

Note: If you do not add the attribute levels, then the levels will be sorted

Ex:

```
fac-val-1 <- factor(val-1 , levels = c("AWD", "4WD", "RWD", "FWD")
print (fac-val-1)
```

```
Output :  AWD   4WD   RWD   FWD
```

Accessing Elements :

Accessing the elements of a factor is same as accessing the elements of a vector.

Modification of a Factor :

After a factor is formed, the new values which need to be assigned must be at the predefined level.

Checking for a Factor :

The function is.factor() is used to check if the variable is a factor and returns TRUE if it is a factor.

R Matrices

In R, matrices are two dimensional homogeneous data structures arranged in rows and columns.

Matrix can be created with the matrix() function. Specify the nrow and ncol parameters to set the amount of rows and columns.

Ex:
```
my-mat <- matrix(c(1:6), nrow = 3,  ncol = 2)
print(my-mat)
```

output :
```
             [,1]  [,2]
      [1,]     1     4
      [2,]     2     5
      [3,]     3     6
```

Accessing Matrix elements:

Elements of the matrix can be accessed using [ ]. The first number specifies the row position and second number specifies the column position.

Ex:
```
print(my-mat[1,2])                 output :  4
```

If you leave the column position empty, the entire row can be accessed. (a comma must be included)

Ex:
```
print(my_mat[1,])                  output :  1  4
```

Similarly the entire column can be accessed by leaving the row position empty.

Ex:
```
print (my-mat [ , 1])
```
output :
```
1 2 3
```

To access more than one row/column use the c( ) function in its position

Ex:
```
my-mat <- matrix (c(1:9) , nrow= 3 , ncol= 3)
print (my-mat [c(1,2) , ])
```

output :
```
1   4   7
2   5   8
```

Add Rows and Columns :

use cbind ( ) and rbind ( ) functions to add a new column or row to the existing matrix.

The cells in the new column/row must be of the same length as the existing matrix.

Ex:
```
new_mat <- rbind( my-mat , c(9 , 3 , 5))
print ( new_mat )
```

output :
```
1   4   7
2   5   8
3   6   9
9   3   5
```

Remove Rows and Columns :

Use -c() function at its corresponding position to remove a row or a column

Ex :

```
new_mat <- new_mat [-c(1) , ]     # this removes the
print (new_mat)                     entire first raw
```

output :

```
2   5   8
3   6   9
9   5   3
```

To get the number of rows and columns in a matrix, use the dim( ) function

Ex :

```
print (dim(new-mat))          output : 3   3
```

R Arrays :

Arrays in R are an essential data storage structures defined by a fixed number of dimensions. They always store homogeneous data.

Uni-dimensional arrays are called vectors (atomic vectors). Two-dimensional arrays are called matrices.

Syntax :

```
variable <- array (data , dim = (nrow, ncol, nmat),
                   dim names = names)
```

nrow = number of rows   ncol = number of columns

nmat = number of matrices of dimensions nrows x ncols

dimnames = names of dimensions (default value = NULL)

```
Ex: my-array <- c(1:24)
    multi_array <- array(my-array, dim = c(4, 3, 2))
```

In the above example, we first create a vector with values from 1 to 24. These values are split into 2 two-dimensional matrices of dimension 4 x 3. ie., the first dimension contains

```
1    5    9
2    6    10
3    7    11
4    8    12
```

the second dimension contains

```
13   17   21
14   18   22
15   19   23
16   20   24
```

```
print (multi_array [2, 3, 2])
```

the above code fetches the element at position 2nd row 3rd column from the second dimension ie., 22

R Strings:

* Strings are a sequence of characters variables. It is a one-dimensional array of characters

* A string in R can be formed by enclosing the group of characters within single quotes or double quotes

Ex:

```
my-str_1 <- 'This is a string!'

my-str-2 <- "This is also a string!!"
```

String Concatenation:

Two or more strings can be concatenated using the paste function.

Syntax: paste(strings, sep = separator , collapse = NULL)

Ex:

```
S1 <- "This"
S2 <- " is "
S3 <- "DAT 430"
S4 <- paste(S1, S2, S3, sep = "  " , collapse = "")
print (S4)
```

Output : This is DAT 430

String length:

Use the inbuit function nchar() to find the number of characters in a string.

Ex:

```
print (nchar(S4))
```

Output : 15

Also, you can find the length of a string using the function str-length() under the stringr package

Ex:
```
library(stringr)
print (str-length(S4))
```

Substring:

A substring is a subset of a larger string.

Syntax:

substring (source-string, start position, end position)

Ex:
```
S5 <-substring(S4, 9, 12)          output : "DAT "
print (S5)
```

Note : The buit-in-functions toupper() and tolower() can be used to convert a string to upper case and lower case respectively.

R Data Frames

* Matrix inputs were limited because all the data inside of the matrix had to be of the same data type.

* A dataframe is a two dimensional data structure which is made up of rows & columns (similar to a matrix)

* Each column of a data frame can be of different types. However, a single column must be of same data type

* Data Frames are data displayed in a format as a table.

Syntax:

```
variable <- data.frame(column vector 1, column vector 2, ...., n)
```

Ex Consider the below table with 5 columns & 4 rows

id	name	age	gender	salary
1	Bob	24	M	17.25
2	Tom	31	M	21.18
3	Alex	25	M	30.7
4	Janice	27	F	34.03

Create 5 vectors to hold the data in 5 columns.

Note: All the columns within a dataframe should be of the exact same length

Pass all the vectors as an argument to data.frame( ) function

Note: Variable names of the vector variables automatically become column name

```
id <- c(1:4)
name <- c("Bob", "Tom", "Alex", "Janice")
age <- c(24, 31, 25, 27)
gender <- c("M", "M", "M", "F")
salary <- c(17.25, 21.18, 30.7, 34.03)

data <- data.frame(id, name, age, gender, salary)
```

Note: By default all atomic character vectors will be treated as factors. To treat an atomic character vector as a string, then set the stringAsFactor parameter to FALSE

* If you want to give custom names to columns.

names(dataFrame) <- vector of column names

Ex:
```
names(data) <- c("employee_id", "employee_name",
"employee_age", "employee_gender", "employee_salary")
```

Accessing Rows:

An entire row can be accessed using the syntax

variable <- dataFrame_name [row_num ,]

Accessing Columns:

Using column numbers -

variable <- dataFrame_name [, col_num]

Using column name

variable <- dataFrame_name $ column_name