KNOWLEDGE ENGINEERING LAB (CSE 4.1.7)

2. Study of homogeneous and heterogeneous data structures such as vector, matrix, array, list, data frame in R.

Data Structure:

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.

R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the five data types which are most frequently utilized in data analysis. the subsequent table shows a transparent cut view of those data structures (1).

Dimension	Homogenous	Heterogeneous
1D	Vector	List
2D	Matrix	Dataframe
nD	Array	

Vector:

A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogenous data structures. Vectors are one-dimensional data structures.

How to create a Vector?

Vectors are generally created using the c() function.

```
> X = c(1, 3, 5, 7, 8)
> print(X)
[1] 1 3 5 7 8
> typeof(X)
[1] "double"
> length(X)
[1] 5
```

```
 > x <- c(1, 5.4, TRUE, "hello")
 > typeof(x)
 [1] "character"
 > length(x)
 [1] 4
 > x
 [1] "1"     "5.4"   "TRUE"  "hello"
 > X
[1] 1 3 5 7 8
```

Creating a vector using operator:

```
 > x <- 1:7
 > x
 [1] 1 2 3 4 5 6 7
 > y <- 2:-2
 > y
 [1]  2  1  0 -1 -2
```

Creating a vector using seq() function:

```
> z <- seq(1, 3, by=0.2)
> z
 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> a<-seq(1, 5, length.out=4)
> a
[1] 1.000000 2.333333 3.666667 5.000000
```

Accessing elements using integer vector as index:

Vector index in R starts from 1, unlike most programming languages where index start from 0. We can use a vector of integers as index to access specific elements. We can also use negative integers to return all elements except that those specified. But we cannot mix positive and neg ative integers while indexing and real numbers, if used, are truncated to integers (2).

```
> x = c(0,  2,  4,  6,  8, 10)
> x
[1]  0  2  4  6  8 10
> x[3]
[1] 4
> x[c(2, 4)]
[1] 2 6
> x[-1]
[1]  2  4  6  8 10          #Access all expect 1st element
```

Accessing elements using logical vector as index:

When we use a logical vector for indexing, the position where the logical vector is TRUE is re turned. This useful feature helps us in filtering of vector as shown below.

```
> x[c(TRUE, FALSE, FALSE, TRUE)]
[1] 0 6 8
> x[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
[1]  0  6  8 10
> x[c(TRUE, FALSE, FALSE)]
[1] 0 6
> x[c(TRUE)]
[1]  0  2  4  6  8 10
> x[c(TRUE, TRUE)]
[1]  0  2  4  6  8 10
> x[c(TRUE, FALSE)]
[1] 0 4 8
> x[x < 0]
numeric(0)
> x[x < 4]
[1] 0 2
> x[x > 4]
[1]  6  8 10
```

Modifying vectors

We can modify a vector using the assignment operator. We can use the techniques discussed a bove to access specific elements and modify them. If we want to truncate the elements, we can use reassignments.

```
> x=c(-3, -2, -1,  0,  1,  2)
> x
[1] -3 -2 -1  0  1  2
> x[2] <- 0
> x
[1] -3  0 -1  0  1  2
> x[x<0] <- 5                    #modify elements less than 0 as 5
> x
[1] 5 0 5 0 1 2
> x <- x[1:4]                    # truncate x to first 4 elements
> x
[1] 5 0 5 0
```

Lists:

A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous da ta structures. These are also one-dimensional data structures. A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on.

Creating Lists

List can be created using the list() function.

```
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
> str(x)
List of 3
 $ a: num 2.5
 $ b: logi TRUE
 $ c: int [1:3] 1 2 3
```

- Structure of the list can be returned using str() function.
- In this example, a, b and c are called tags which makes it easier to reference the componen ts of the list. However, tags are optional. We can create the same list without the tags as fol lows. In such scenario, numeric indices are used by default.

```
> x <- list(2.5,TRUE,1:3)
> x
[[1]]
[1] 2.5

[[2]]
[1] TRUE

[[3]]
[1] 1 2 3


> empId = c(1, 2, 3, 4)
> empName = c("Debi", "Sandeep", "Subham", "Shiba")
> numberOfEmp = 4
> empList = list(empId, empName, numberOfEmp)
> print(empList)
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Debi"    "Sandeep" "Subham"  "Shiba"

[[3]]
[1] 4
```

Name List Elements in R Language

```
> data_list <- list(c("Jan","Feb","Mar"), matrix(c(1,2,3,4,-1,9), nrow = 2)
,list("Red",12.3))
> data_list
[[1]]
[1] "Jan" "Feb" "Mar"

[[2]]
     [,1] [,2] [,3]
[1,]    1    3   -1
[2,]    2    4    9

[[3]]
[[3]][[1]]
[1] "Red"

[[3]][[2]]
[1] 12.3


> names(data_list) <- c("Monat", "Matrix", "Misc")
> data_list
$Monat
[1] "Jan" "Feb" "Mar"

$Matrix
     [,1] [,2] [,3]
[1,]    1    3   -1
[2,]    2    4    9

$Misc
$Misc[[1]]
[1] "Red"

$Misc[[2]]
[1] 12.3
```

Accessing Elements from the Lists (3)

- Accessing elements by index

```
> print(data_list[3])
$Misc
```

```
$Misc[[1]]
[1] "Red"

$Misc[[2]]
[1] 12.3

> print(data_list[1])
$Monat
[1] "Jan" "Feb" "Mar"
```

- Accessing elements by name

```
> print(data_list$Matrix)
     [,1] [,2] [,3]
[1,]    1    3   -1
[2,]    2    4    9
> print(data_list$Monat)
[1] "Jan" "Feb" "Mar"
```

- Merging Lists

```
> num_list <- list(1,2,3,4,5)
> day_list <- list("Mon","Tue","Wed", "Thurs", "Fri")
> merge_list <- c(num_list, day_list)
> merge_list
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 4

[[5]]
[1] 5

[[6]]
[1] "Mon"

[[7]]
[1] "Tue"
```

```
[[8]]
[1] "Wed"

[[9]]
[1] "Thurs"

[[10]]
[1] "Fri"
```

Matrices:

A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are t wo-dimensional, homogeneous data structures.

Now, let's see how to create a matrix in R. To create a matrix in R you need to use the function c alled matrix. The arguments to this matrix() are the set of elements in the vector. You have to pas s how many numbers of rows and how many numbers of columns you want to have in your matri x and this is the important point you have to remember that by default, matrices are in column-wi se order.

```
> A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE
)
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> matrix(1:9, nrow = 3, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Accessing elements of a matrix (4)

```
> A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> A[c(1,2),c(2,3)]          # select rows 1 & 2 and columns 2 & 3
```

```
     [,1] [,2]
[1,]    4    7
[2,]    5    8

> A[c(3,2),]   # leaving column field blank will select entire columns
     [,1] [,2] [,3]
[1,]    3    6    9
[2,]    2    5    8

> A[c(TRUE,FALSE,TRUE),c(TRUE,TRUE,FALSE)]
     [,1] [,2]
[1,]    1    4
[2,]    3    6

> A[A>5]
[1] 6 7 8 9


> x <- matrix(1:9, nrow = 3, dimnames = list(c("X","Y","Z"), c("A","B","C")
))
> x
  A B C
X 1 4 7
Y 2 5 8
Z 3 6 9
> x[,"A"]
X Y Z
1 2 3
> x[TRUE,c("A","C")]
  A C
X 1 7
Y 2 8
Z 3 9
> x[2:3,c("A","C")]
  A C
Y 2 8
Z 3 9
```

Modifying elements of a matrix

```
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
[3,]    3    6    9
> A[2,2] <- 10
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2   10    8
[3,]    3    6    9
> A[A<5] <- 0
> A
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
[3,]    0    6    9
```

Dataframes:

Dataframes are generic data objects of R which are used to store the tabular data. Dataframes are the foremost popular data objects in R programming because we are comfortable in seeing the da ta within the tabular form. They are two-dimensional, heterogeneous data structures. These are lis ts of vectors of equal lengths (5).

Data frames have the following constraints placed upon them:

- A data-frame must have column names.
- Each column must have the identical number of items.
- Each item in a single column must be of the same data type.
- Different columns may have different data types.

To create a dataframe we use the data.frame() function.

```
> Name = c("Amiya", "Raj", "Asish")
> Language = c("R", "Python", "Java")
> Age = c(22, 25, 45)
> df = data.frame(Name, Language, Age)
> print(df)
   Name Language Age
1 Amiya        R  22
2   Raj   Python  25
3 Asish     Java  45
> names(df)
[1] "Name"     "Language" "Age"
> ncol(df)
[1] 3
```

```
> nrow(df)
[1] 3

> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))
> str(x)
'data.frame':       2 obs. of  3 variables:
 $ SN  : int  1 2
 $ Age : num  21 15
 $ Name: Factor w/ 2 levels "Dora","John": 2 1
```

Accessing components of dataframe:

We can use either [, [[or $ operator to access columns of data frame

```
> x["Name"]
  Name
1 John
2 Dora
> x$Name
[1] John Dora
> x$Name
[1] John Dora
> x[["Name"]]
[1] John Dora
> x[[3]]
[1] John Dora
```

```
> trees <- data.frame("Girth" = c(8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11, 11,
11.1, 11.2), "Height" = c(70, 65, 63, 72, 81, 83, 66, 75, 80, 75), "Volume"
= c(10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6, 18.2, 22.6, 19.9))
> str(trees)
'data.frame':       10 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
> head(trees,n=3)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
> trees[2:3,]
  Girth Height Volume
2   8.6     65   10.3
```

```
3   8.8     63   10.2
> trees[3:3,]
  Girth Height Volume
3   8.8     63   10.2
> trees[2:5,]
  Girth Height Volume
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
> trees[trees$Height > 82,]
  Girth Height Volume
6  10.8     83   19.7
> trees[trees$Height > 80,]
  Girth Height Volume
5  10.7     81   18.8
6  10.8     83   19.7
```

Modifying elements of dataframe

```
x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))
> x
  SN Age Name
1  1  21 John
2  2  15 Dora
> x[1,"Age"] <- 20
> x
  SN Age Name
1  1  20 John
2  2  15 Dora
```

Arrays:

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dim ensional data structures. For example, if we create an array of dimensions (2, 3, 3) then it creates 3 rectangular matrices each with 2 rows and 3 columns. They are homogeneous data structures. Now, let's see how to create arrays in R. To create an array in R you need to use the function call ed array(). The arguments to this array() are the set of elements in vectors and you have to pass a vector containing the dimensions of the array.

```
> A = array(c(1, 2, 3, 4, 5, 6, 7, 8), dim = c(2, 2, 2))
> A
, , 1
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8



> arr1 <- array(c(1:18),dim=c(2,3,3))
> arr1
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18



> rnames <- c("r1","r2","r3")
> cnames <- c("c1","c2","c3")
> mnames <- c("m1","m2","m3")
> named_array <- array(c(1:27),dim=c(3,3,3),dimnames= list(rnames,cnames,mn
ames))
> named_array
, , m1

   c1 c2 c3
```

```
r1   1   4   7
r2   2   5   8
r3   3   6   9

, , m2

   c1 c2 c3
r1 10 13 16
r2 11 14 17
r3 12 15 18

, , m3

   c1 c2 c3
r1 19 22 25
r2 20 23 26
r3 21 24 27
```

<u>Acessing elements from arrays</u>

```
> named_array[2,1,3]        #second row, first column and third matrix of named_array
[1] 20

> named_array[c(1,2),c(2,3),c(1,2)]
#first and second row, second and third column of first and second matrices
, , m1

   c2 c3
r1  4  7
r2  5  8

, , m2

   c2 c3
r1 13 16
r2 14 17


> named_array[-1,-3,-2]
#remove first row, third column and second matrix of named_array
, , m1

   c1 c2
r2  2  5
```

```
r3  3  6

, , m3

   c1 c2
r2 20 23
r3 21 24
```

```
> named_array[c(T,F,T),c(F,T,T),c(T,T,F)]
#first and third row, second and third column of first and second matrix of
named_array
, , m1

   c2 c3
r1  4  7
r3  6  9

, , m2

   c2 c3
r1 13 16
r3 15 18
```

Modifying an array

We can use the indexing techniques to access elements or parts of an array. Then we can use r e-assignment to change their values. For example:

```
> named_array[1,,1] <- c(1,2,3)
# 1st row of the 1st matrix is replaced with 1, 2, 3
> named_array
, , m1

   c1 c2 c3
r1  1  2  3
r2  4  4  4
r3  3  6  9

, , m2

   c1 c2 c3
r1 10 13 16
r2 11 14 17
```

```
r3 12 15 18

, , m3

   c1 c2 c3
r1 19 22 25
r2 20 23 26
r3 21 24 27
```

Array Arithmetic

```
> test_arr1 <- array(c(1:8),c(2,2,2))
> test_arr2 <- array(c(9:16),c(2,2,2))
> arr_add <- test_arr1+test_arr2
> arr_add
, , 1

     [,1] [,2]
[1,]   10   14
[2,]   12   16

, , 2

     [,1] [,2]
[1,]   18   22
[2,]   20   24
> arr_sub <- test_arr2-test_arr1
> arr_sub
, , 1

     [,1] [,2]
[1,]    8    8
[2,]    8    8

, , 2

     [,1] [,2]
[1,]    8    8
[2,]    8    8
> arr_mul <- test_arr1*test_arr2
> arr_mul
, , 1

     [,1] [,2]
[1,]    9   33
```

```
[2,]   20   48

, , 2

     [,1] [,2]
[1,]   65  105
[2,]   84  128

> arr_div <- test_arr2/test_arr1
> arr_div
, , 1

     [,1]     [,2]
[1,]    9 3.666667
[2,]    5 3.000000

, , 2

         [,1]     [,2]
[1,] 2.600000 2.142857
[2,] 2.333333 2.000000

> arr_pow <- test_arr1^2
> arr_pow
, , 1

     [,1] [,2]
[1,]    1    9
[2,]    4   16

, , 2

     [,1] [,2]
[1,]   25   49
[2,]   36   64
```

References

1. Data Structures in R Programming [Internet]. Available from: https://www.geeksforgeeks.org/data-structures-in-r-programming/#:~:text=Arrays are the R data,They are homogeneous data structures.
2. R Vector [Internet]. Available from: https://www.datamentor.io/r-programming/vector/

3. DataFlair. R List – Learn what all you can do with Lists in R! [Internet]. Available from: https://data-flair.training/blogs/r-list-tutorial/
4. R Matrix [Internet]. Available from: https://www.datamentor.io/r-programming/matrix/
5. R Data Frame [Internet]. Available from: https://www.datamentor.io/r-programming/data-frame/