UNIT-V

STRINGS:

Any value coded within a pair of single quote (' ') or double quotes (" ") in R is treated as a String. R stores all the strings within double quotes, even if you use single quotes to create them. A string is declared as shown below.

```
str <- 'Hello World'
print(str)
[1] "Hello Word"
```

```
str <- "Hello World"
print(str)
[1] "Hello Word"
```

We can perform the following operations on strings.

1. Concatenation
2. Character count
3. Change case
4. Sub string

We can perform all these operations on strings using predefined functions associated with strings.

Concatenation:

Concatenation represents adding up of two strings. We can perform concatenation using "pase( )" function.

paste( ) function takes two strings and append second string at the end of first string.

```
str1<- 'Hi'
str2<- 'How are you?'
paste(str1,str2)
print(str1)
[1] "Hi How are you?"
```

Character Count:

We count the number of characters in a given string using "nchar( )" function. nchar( ) function takes a string as parameter and return an integer value.

```
str1 <- "aditya"
nchar(str1)
[1] 6
```

Change Case:

Change case operation performs conversion of given string into its toggle case. Change case operation is performed using two functions.

1. toupper( ) – This function is used convert given string into upper case characters if it was in lower case characters.
2. tolower( )- This function is used to convert given string into upper case characters if it was in upper case characters.

Scanned with CamScanner

toupper( )	tolower( )
str <- "Aditya"	str <- "Aditya"
toupper(str)	tolower(str)
print(str)	print(str)
[1] "ADITYA"	[1] "aditya"

Substring:

Substrig operation is performed using substring( ) function.

substring( ) function takes three parameters.

1. String
2. Start index
3. End index

The substring( ) function returns a substring starting from start index upto end index.

```
str <- "programming"
substring(str,4,7)
[1] "gram"
```

VECTORS

Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.

A vector's type can be checked with the typeof( ) function or class( ) function.

Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function length( ).

Vectors are generally created using the c( ) function.

Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.

Coercion is from lower to higher types from logical to integer to double to character.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> length(x)
[1] 5
> x <- c(1, 5.4, TRUE, "hello")
> x
[1] "1"     "5.4"   "TRUE"  "hello"
> typeof(x)
[1] "character"
```

If we want to create a vector of consecutive numbers, the : operator is very helpful.

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

Creating Complex vectors:

More complex sequences can be created using the seq( ) function, like defining number of points in an interval, or the step size.

```
> seq(1, 3, by=0.2)          # specify step size
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> seq(1, 5, length.out=4)    # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
```

Accessing Vectors:

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

Scanned with CamScanner

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 negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1] 0 2 4 6 8 10
> x[3]          # access 3rd element
[1] 4
> x[c(2, 4)]    # access 2nd and 4th element
[1] 2 6
> x[-1]         # access all but 1st element
[1] 2 4 6 8 10
> x[c(2, -4)]   # cannot mix positive and negative integers
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
> x[c(2.4, 3.54)]   # real numbers are truncated to integers
[1] 2 4
```

Modifying vectors:

We can modify a vector using the assignment operator.

We can use the techniques discussed above to access specific elements and modify them.

If we want to truncate the elements, we can use reassignments.

```
> x
[1] -3 -2 -1 0 1 2
> x[2] <- 0; x       # modify 2nd element
[1] -3 0 -1 0 1 2
> x[x<0] <- 5; x   # modify elements less than 0
[1] 5 0 5 0 1 2
> x <- x[1:4]; x     # truncate x to first 4 elements
[1] 5 0 5 0
```

Deleting Vectors:

We can delete a vector by simply assigning a NULL to it.

```
> x
[1] -3 -2 -1 0 1 2
> x <- NULL
> x
NULL
> x[4]
NULL
```

LISTS

List is a data structure having components of mixed data types.

A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.

We can check if it's a list with typeof() function and find its length using length(). Here is an example of a list having three components each of different data type.

List can be created using the list() function.

Scanned with CamScanner

```
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
```

Its structure can be examined with the str() function.

```
> str(x)
List of 3
$ a: num 2.5
$ b: logi TRUE
$ c: int [1:3] 1 2 3
```

In this example, a, b and c are called tags which makes it easier to reference the components of the list. However, tags are optional. We can create the same list without the tags as follows. 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
```

Accessing List Components:

Lists can be accessed in similar fashion to vectors. Integer, logical or character vectors can be used for indexing. Let us consider a list as follows.

```
> x
$name
[1] "John"
$age
[1] 19
$speaks
[1] "English" "French"
> x[c(1:2)]    # index using integer vector
$name
[1] "John"
$age
[1] 19
> x[-2]        # using negative integer to exclude second component
$name
[1] "John"
$speaks
[1] "English" "French"
> x[c(T,F,F)]  # index using logical vector
$name
[1] "John"
> x[c("age","speaks")]    # index using character vector
$age
[1] 19
$speaks
[1] "English" "French"
```

Scanned with CamScanner

Modifying lists:

We can change components of a list through reassignment. We can choose any of the component accessing techniques discussed above to modify it.

```
> x[["name"]] <- "Clair"; x
$age
[1] 19
$speaks
[1] "English" "French"
$name
[1] "Clair"
```

MATRICES

Matrix is a two dimensional data structure in R programming.

Matrix is similar to vector but additionally contains the dimension attribute. All attributes of an object can be checked with the attributes() function (dimension can be checked directly with the dim() function).

We can check if a variable is a matrix or not with the class() function.

Matrix can be created using the matrix() function.

Dimension of the matrix can be defined by passing appropriate value for arguments nrow and ncol.

Providing value for both dimension is not necessary. If one of the dimension is provided, the other is inferred from length of the data.

```
> matrix(1:9, nrow = 3, ncol = 3)
[,1] [,2] [,3]
[1,]   1   4   7
[2,]   2   5   8
[3,]   3   6   9
> # same result is obtained by providing only one dimension
> matrix(1:9, nrow = 3)
[,1] [,2] [,3]
[1,]   1   4   7
[2,]   2   5   8
[3,]   3   6   9
```

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing TRUE to the argument byrow.

```
> matrix(1:9, nrow=3, byrow=TRUE)   # fill matrix row-wise
[,1] [,2] [,3]
[1,]   1   2   3
[2,]   4   5   6
[3,]   7   8   9
```

It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument dimnames.

```
> 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
```

Scanned with CamScanner

Accessing Matrix Elements:

We can access elements of a matrix using the square bracket [indexing method. Elements can be as var[row, column]. Here rows and columns are vectors.

We specify the row numbers and column numbers as vectors and use it for indexing.

If any field inside the bracket is left blank, it selects all.

We can use negative integers to specify rows or columns to be excluded.

```
> x
[,1] [,2] [,3]
[1,]   1   4   7
[2,]   2   5   8
[3,]   3   6   9
> x[c(1,2),c(2,3)]   # select rows 1 & 2 and columns 2 & 3
[,1] [,2]
[1,]   4   7
[2,]   5   8
> x[c(3,2),]   # leaving column field blank will select entire columns
[,1] [,2] [,3]
[1,]   3   6   9
[2,]   2   5   8
> x[,]   # leaving row as well as column field blank will select entire matrix
[,1] [,2] [,3]
[1,]   1   4   7
[2,]   2   5   8
[3,]   3   6   9
> x[-1,]   # select all rows except first
[,1] [,2] [,3]
[1,]   2   5   8
[2,]   3   6   9
```

DATA FRAMES

Data frame is a two dimensional data structure in R. It is a special case of a list which has each component of equal length.

Each component form the column and contents of the component form the rows.

We can create a data frame using the data.frame() function.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))
> str(x)   # structure of 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
```

Many data input functions of R like, read.table(), read.csv(), read.delim(), read.fwf() also read data into a data frame.

Components of data frame can be accessed like a list or like a matrix.

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

Scanned with CamScanner

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

Data frames can be modified like we modified matrices through reassignment.

```
> 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
```

(5) RESHAPING

1. Data Reshaping in R is about changing the way data is organized into rows and columns.
2. Most of the time data processing in R is done by taking the input data as a data frame.
3. It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it.
4. R has many functions to split, merge and change the rows to columns and vice-versa in a data frame.
5. We use cast( ) function and melt( ) function to perform reshaping in R.

(6) PACKAGES

- A package is a set of R functions and data-sets and the library is a folder on your system / computer which stores the files for those package(s).
- For loading a package which is already existing and installed on your system, you can make use of and call the library function.
- It is to be noted that the name of the package needs to be passed to library without being enclosed within quotes.

```
pkges <- c ("lattice", "utils", "rpart")
for (pkg1 in pkges)
{
library (pkg1, character.only = TRUE)
}
```

```
> library()
```

When you execute the above code snippet, it generates the below mentioned output.

Scanned with CamScanner

```
Packages in library 'C:/Program Files/R/R-3.2.2/library':

base            The R Base Package
boot            Bootstrap Functions (Originally by Angelo Canty
                for S)
class           Functions for Classification
cluster         "Finding Groups in Data": Cluster Analysis
                Extended Rousseeuw et al.
codetools       Code Analysis Tools for R
compiler        The R Compiler Package
```

Installing a package

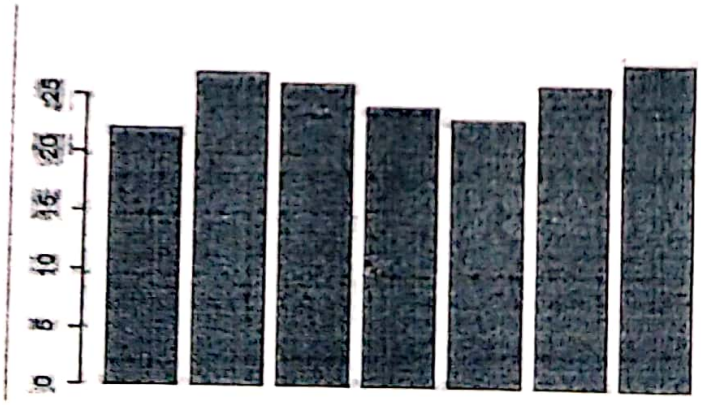
```
install. packages("Package Name")
# Install the package named "XML".
install. packages ("XML")
```

BAR GRAPH

Bar plots can be created in R using the barplot() function. We can supply a vector or matrix to this function. If we supply a vector, the plot will have bars with their heights equal to the elements in the vector.

```
max.temp <- c(22, 27, 26, 24, 23, 26, 28)

barplot(max.temp)
```



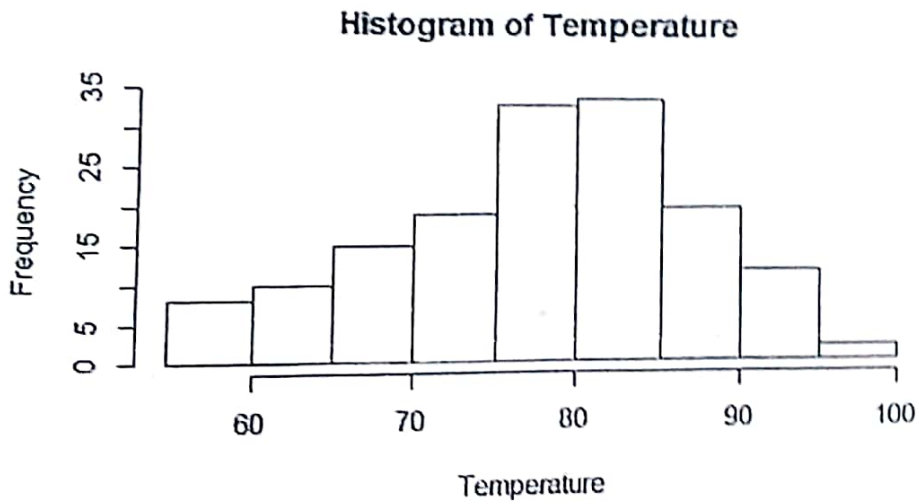
HISTOGRAM

Histogram can be created using the hist() function in R programming language. This function takes in a vector of values for which the histogram is plotted.

```
> str(airquality)
'data.frame':    153 obs. of  6 variables:
$ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
$ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
$ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
$ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
$ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
$ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Scanned with CamScanner

```
Temperature <- airquality$Temp
hist(Temperature)
```



Scanned with CamScanner