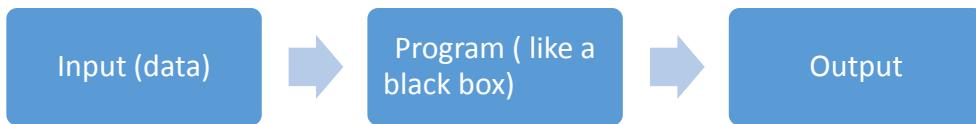


Basic R Programming

1. Program

- It is required have some data / input for the program.
- Input is set of instructions to work on the data.



2. Variable (Objects)

- A variable is storage for values inside R-Object that our programs can manipulate.
- A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number. For Example: var1, var_1, .var1, vAr1.
- The variables can be assigned values using leftward, rightward and equal to operator.

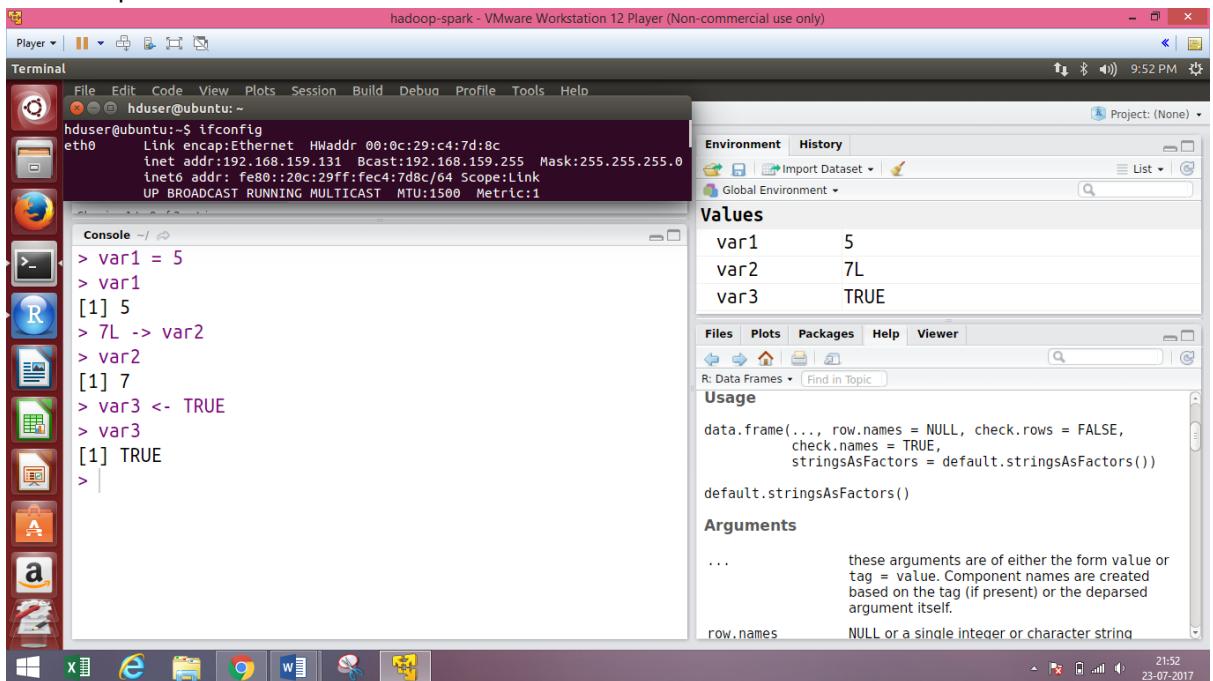


Figure 1: Variable Assignment

- These values can be Character, Numeric, Integer, Complex, Logical and Raw **Data Type**. These are called as **Class or Mode**. These data can also be stored in complex – atomic vector variables such as Vectors, Lists, Matrices, Arrays, Factors and Data Frames.
 - i. **Logical:** It can be either TRUE or FALSE

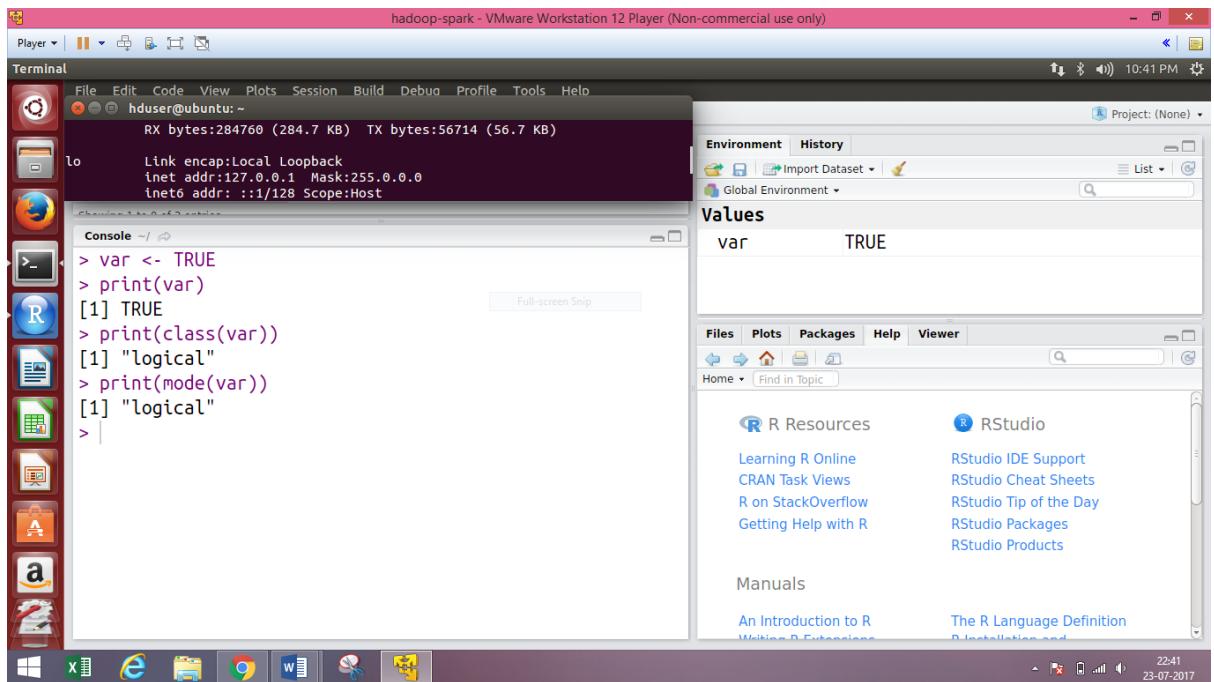


Figure 2: Logical Object

ii. **Numeric:** It is any numeric number. It is root class for Integer and Decimal numbers.

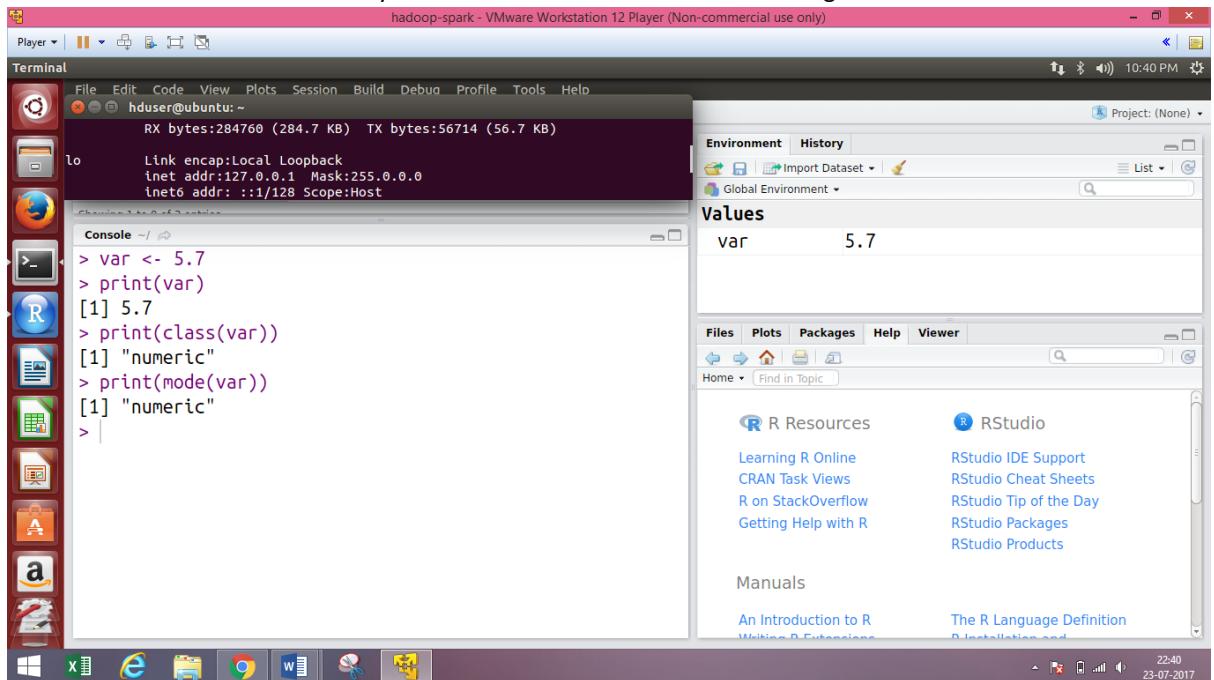


Figure 3: Numeric Object

iii. **Integer:** It is strictly whole number. This can be positive, negative or zero. It is subclass of Numeric class. It is initiated with number preceding to L. For example: 5L

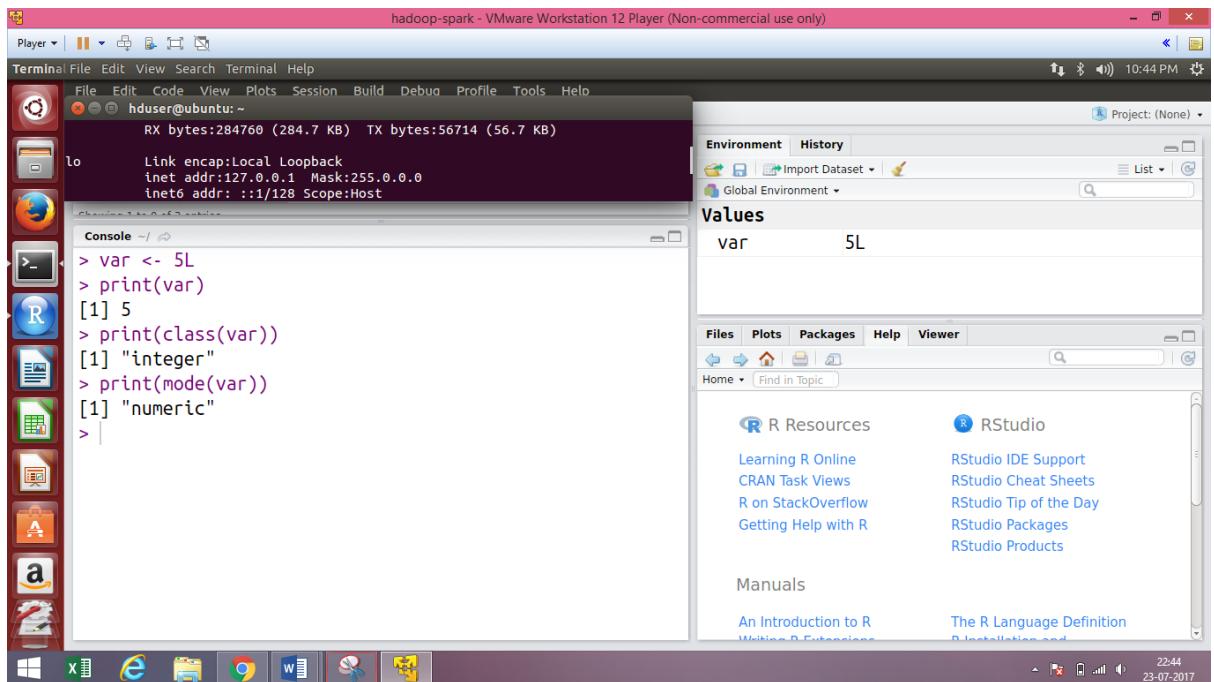


Figure 4: Integer Object

iv. **Complex**: It is complex numeric number. For example: $5 + 7i$

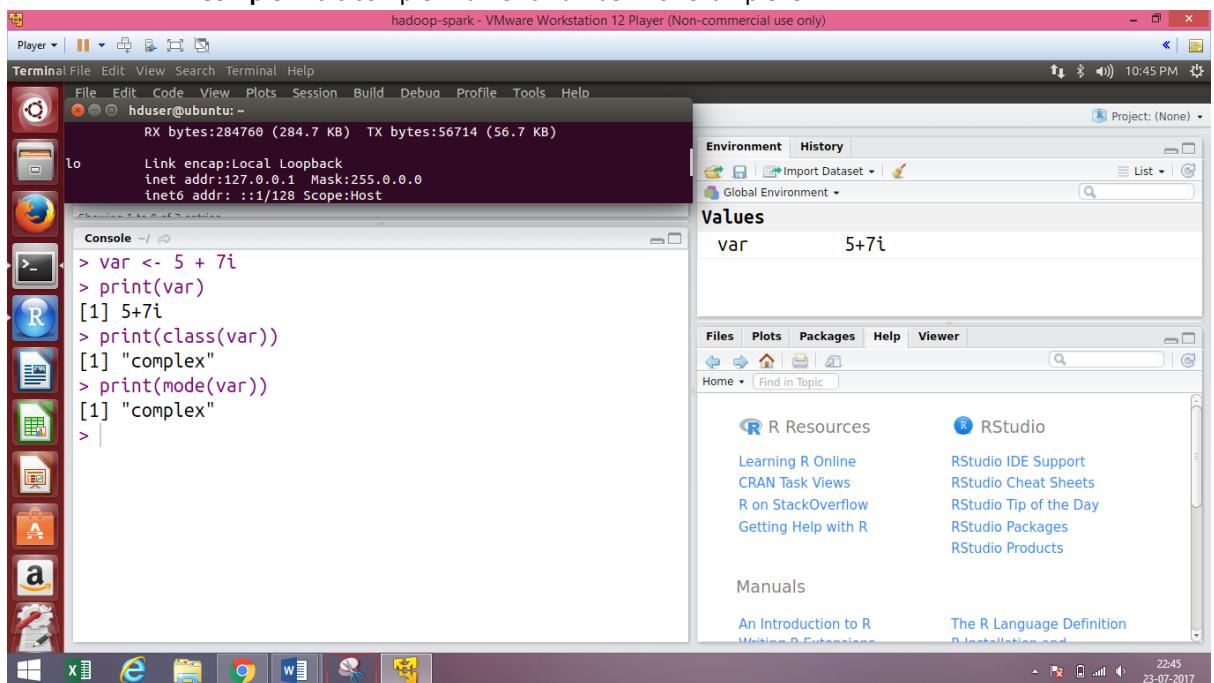


Figure 5: Complex Object

v. **Character**: It is any character or string variable.

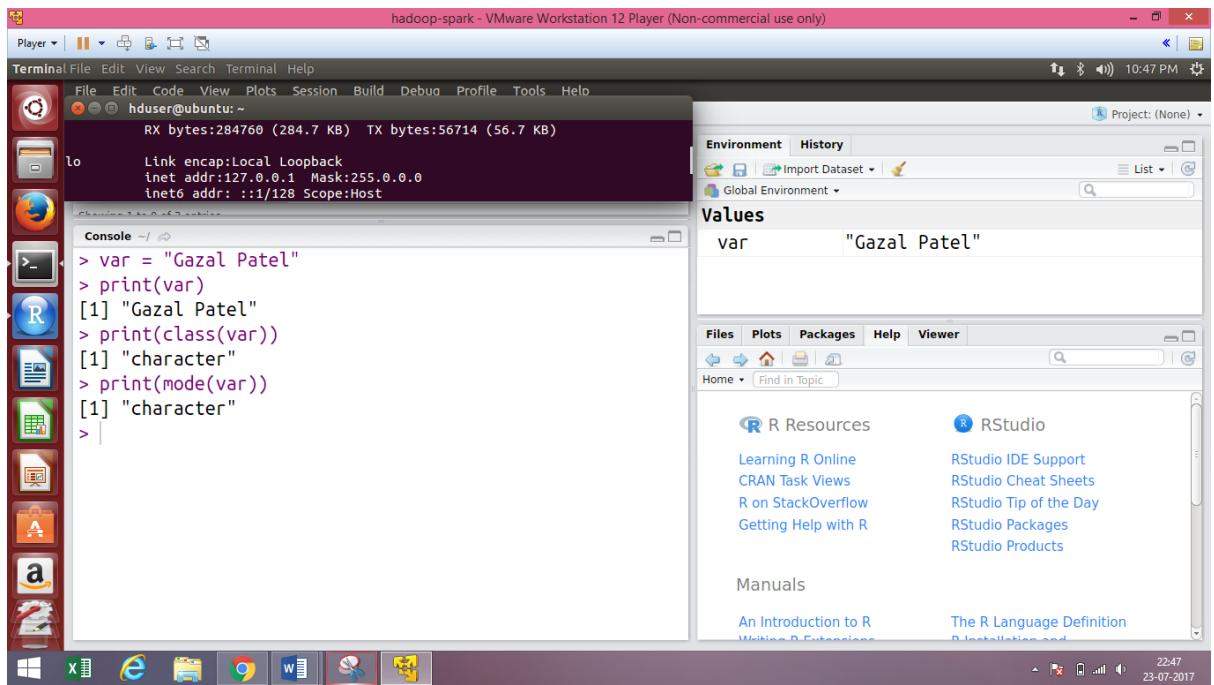


Figure 6: Character Object

- vi. **Raw:** The raw type is intended to hold raw bytes. It is possible to extract subsequences of bytes, and to replace elements (but only by elements of a raw vector). A raw vector is printed with each byte separately represented as a pair of hex digits. Coercion to raw treats the input values as representing small (decimal) integers, so the input is first coerced to integer, and then values which are outside the range [0 ... 255] or are NA are set to 0 (the **nul** byte). For Example, "Hello" is stored as 48 65 6c 6c 6f

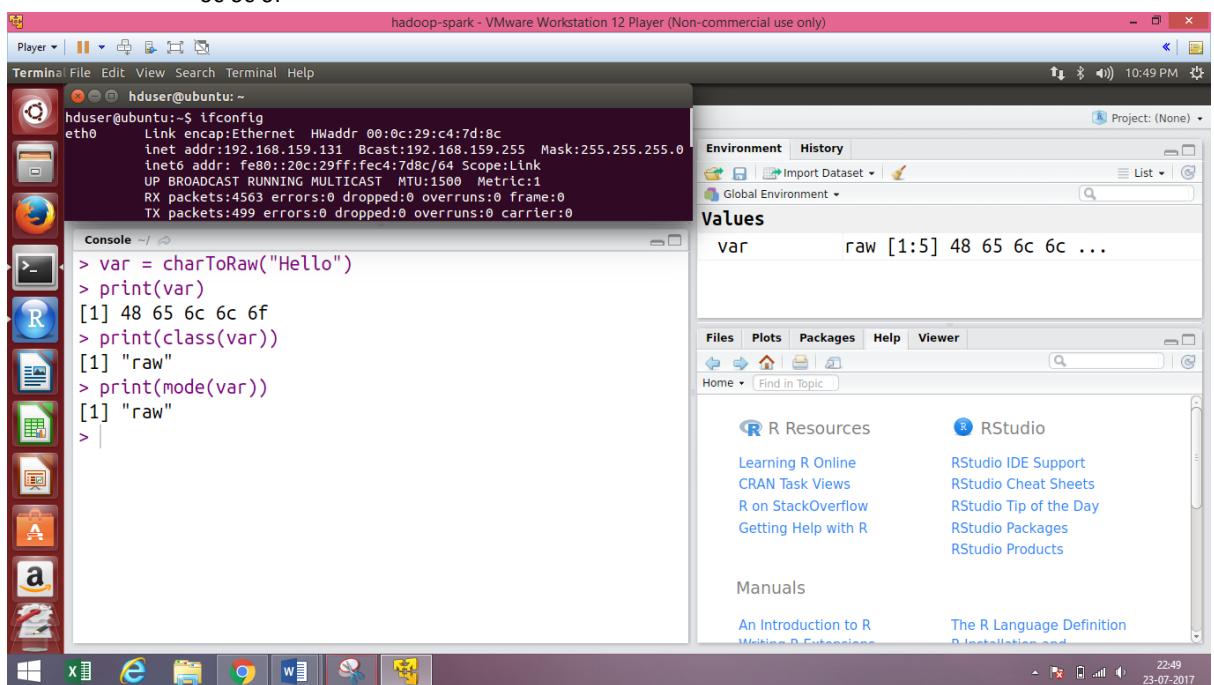


Figure 7: Raw Object

- vii. **Vectors:** It is collection of more than one data. Data can be of any datatype. Use **c()** function to create vector. Also See [SEQ](#)

Basic Programming Assignment-2

Gazal Patel - DSFT1701173

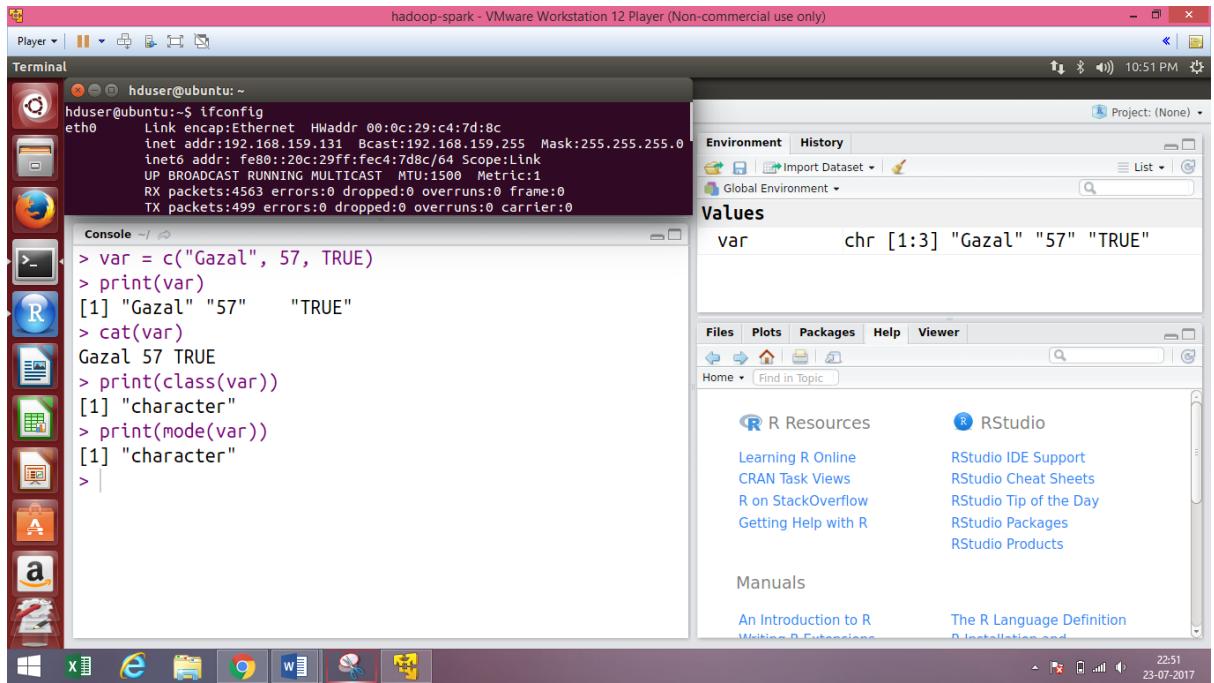


Figure 8: Vector

- viii. **Lists:** A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

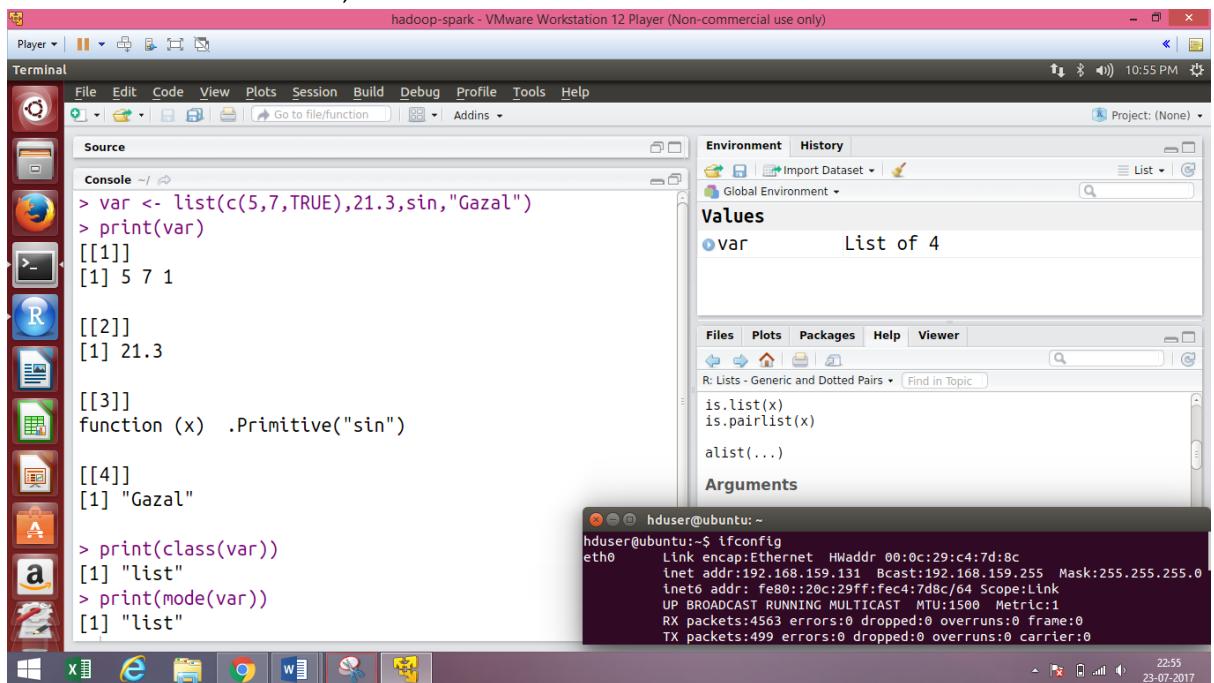


Figure 9: List

- ix. **Matrix:** A matrix is a two-dimensional rectangular data set. It is confined to two dimensions. It can be created using a vector input to the `matrix()` function.
Also see [Difference with Array](#).

The screenshot shows the RStudio interface with the following code in the console:

```
> var = matrix(c('g','a','z','a','l','p','a','t','e','l'), nrow = 2, ncol = 5, byrow = TRUE)
> print(var)
[,1] [,2] [,3] [,4] [,5]
[1,] "g"   "a"   "z"   "a"   "l"
[2,] "p"   "a"   "t"   "e"   "l"
> print(class(var))
[1] "matrix"
> print(mode(var))
[1] "character"
> var = matrix(c('g','a','z',5:7), nrow = 2, ncol = 3, byrow = TRUE)
> print(var)
[,1] [,2] [,3]
[1,] "g"   "a"   "z"
[2,] "5"   "6"   "7"
> print(class(var))
[1] "matrix"
> print(mode(var))
[1] "character"
```

Below the console, a terminal window shows the output of the command `tconfig`:

```
hduser@ubuntu:~$ tconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:4563 errors:0 dropped:0 overruns:0 frame:0
            TX packets:499 errors:0 dropped:0 overruns:0 carrier:0
```

Figure 10: Matrix

- x. **Array:** Arrays can be seen as matrix but consisting of any number of dimensions. The **array()** function takes a 'dim' attribute which creates the required number of dimension. Attribute 'dim' holds vector of three: total rows, columns and repetition of data-vector to fill array.
- Difference with Matrix here is that in matrix number of data elements to be filled should be multiple of rows and columns number. On the other hand array does repetition of data until for required data space.

The screenshot shows the RStudio interface with the following code in the console:

```
> var = array(c('g','b'),dim = c(2,3,2))
> print(var)
, , 1
[,1] [,2] [,3]
[1,] "g"   "g"   "g"
[2,] "b"   "b"   "b"
, , 2
[,1] [,2] [,3]
[1,] "g"   "g"   "g"
[2,] "b"   "b"   "b"
> print(class(var))
[1] "array"
> print(mode(var))
[1] "character"
```

Below the console, a terminal window shows the output of the command `tconfig`:

```
hduser@ubuntu:~$ tconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:4563 errors:0 dropped:0 overruns:0 frame:0
            TX packets:499 errors:0 dropped:0 overruns:0 carrier:0
```

Figure 11: Arrays

The screenshot shows the RStudio interface with the following code in the Console:

```
> var = array(c('g','b'),dim = c(3,3,2))
> print(var)
, , 1

[,1] [,2] [,3]
[1,] "g"   "b"   "g"
[2,] "b"   "g"   "b"
[3,] "g"   "b"   "g"

, , 2

[,1] [,2] [,3]
[1,] "b"   "g"   "b"
[2,] "g"   "b"   "g"
[3,] "b"   "g"   "b"
>
```

The RStudio interface includes a sidebar with icons for various tools like R, browser, and file explorer, and a right-hand panel titled "R Resources" with links to R documentation and support.

Figure 12: Array Explained

- xi. **Factor:** It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. It is created using the **factor()** function. The **nlevels()** functions gives the count of levels.

The screenshot shows the RStudio interface with the following code in the Console:

```
> var = c('Cold','Warm','Warm','Hot','Cold','Hot','Hot','Cold')
> print(var)
[1] "Cold" "Warm" "Warm" "Hot" "Cold" "Hot" "Hot" "Cold"
> factor_var = factor(var)
> print(factor_var)
[1] Cold Warm Warm Hot Cold Hot Hot Cold
Levels: Cold Hot Warm
> print(nlevels(factor_var))
[1] 3
> print(class(factor_var))
[1] "factor"
> print(mode(factor_var))
[1] "numeric"
>
```

Below the RStudio window, a terminal window is open showing the command `ifconfig` and its output:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:4563 errors:0 dropped:0 overruns:0 frame:0
            TX packets:499 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:400000 (389.1 KB) TX bytes:80000 (78.6 KB)
```

Figure 13: Factor

- xii. **Data Frame:** Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length. Data Frames are created using the **data.frame()** function.

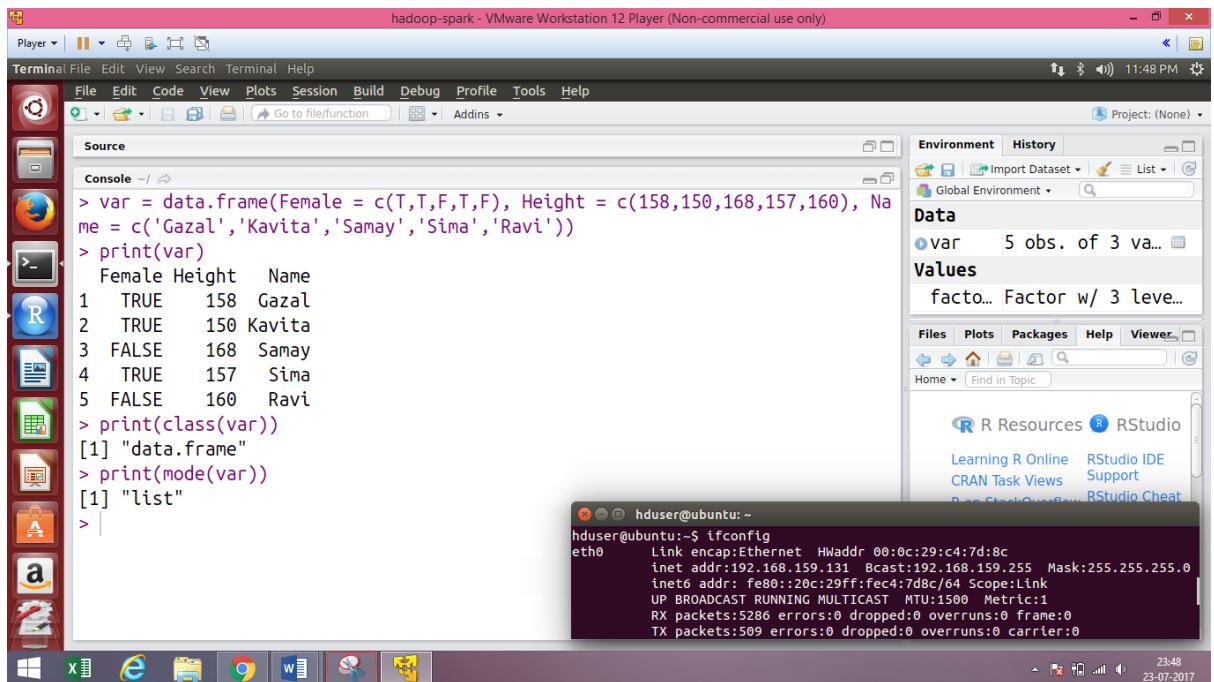


Figure 14: Data Frame

- Printing Variable can be done by **print()** or **cat()** functions.

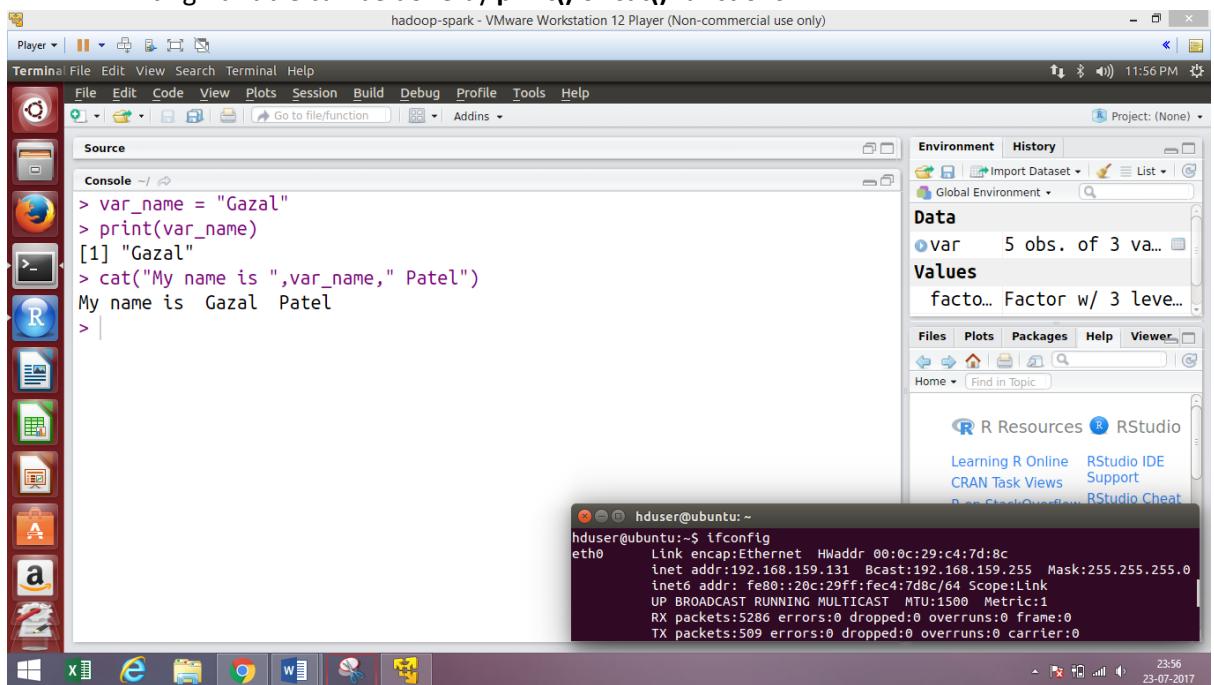


Figure 15: print() and cat()

- Search Variables

- ls()** : Print all currently available variables of the current workspace

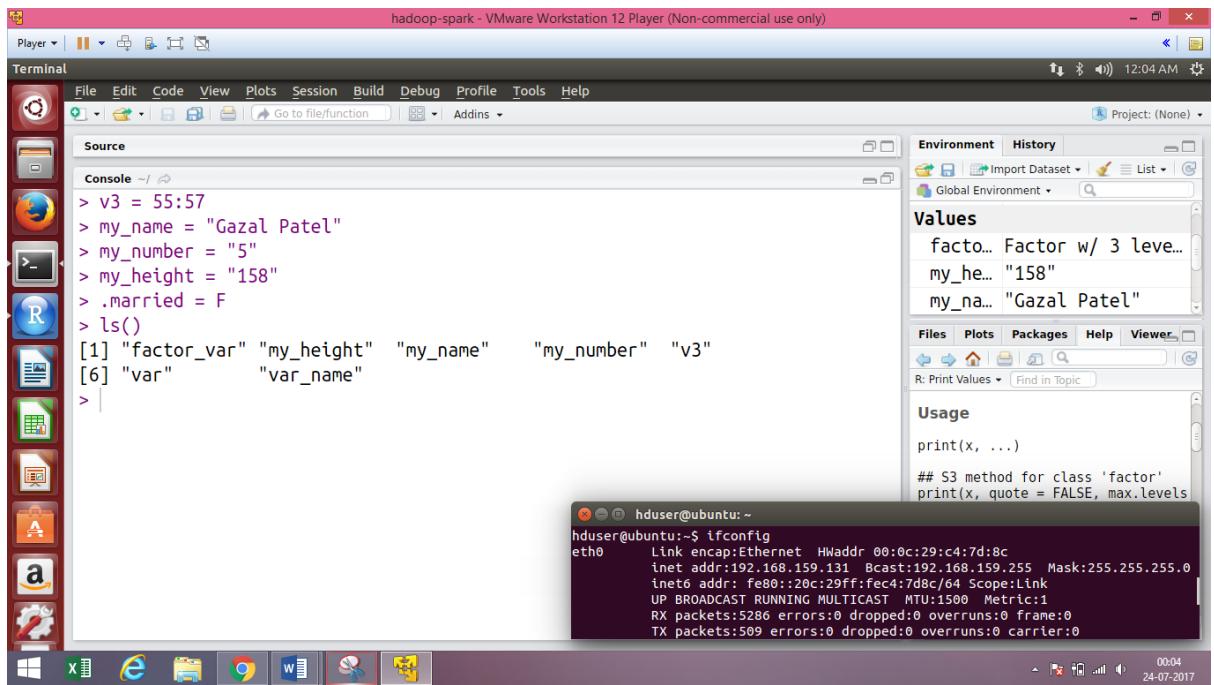


Figure 16: `ls()`

ii. **`ls()` for pattern :** `ls()` function can use patterns to match the variable names.

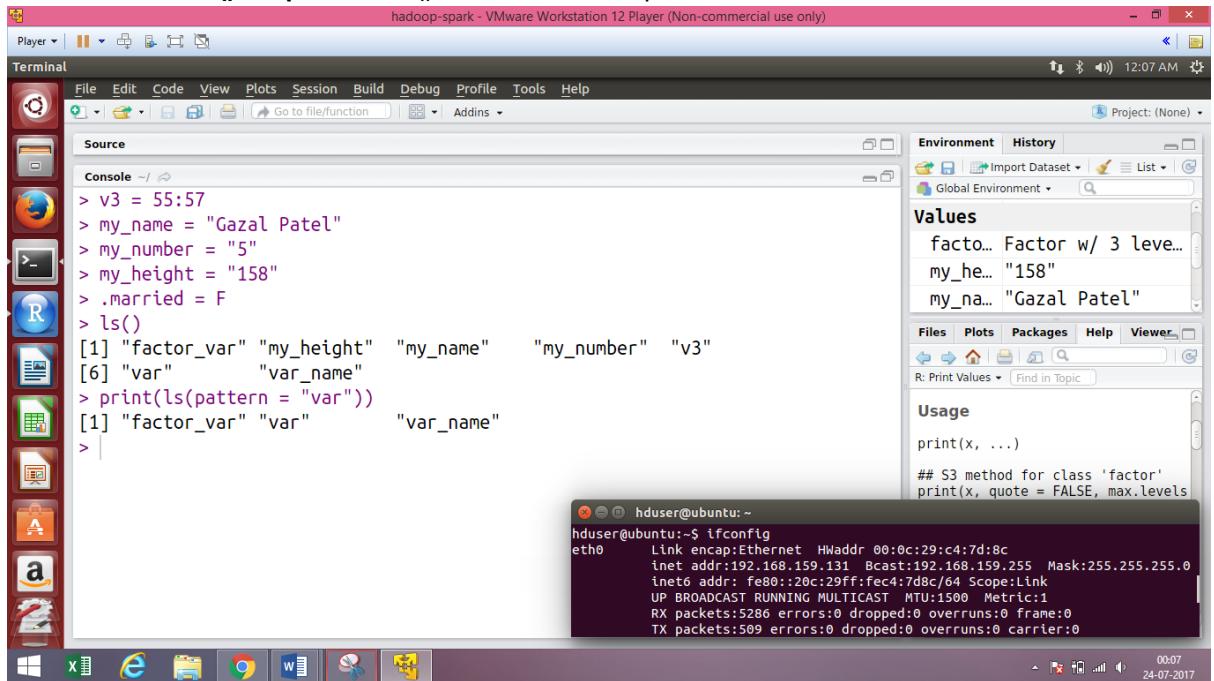


Figure 17: `ls()` for Pattern

iii. **`ls()` for Hidden Variable :** The variables starting with dot(.) are hidden, they can be listed using "all.names = TRUE" argument to `ls()` function.

The screenshot shows the RStudio interface with the following console output:

```

> v3 = 55:57
> my_name = "Gazal Patel"
> my_number = "5"
> my_height = "158"
> .married = F
> ls()
[1] "factor_var" "my_height"   "my_name"      "my_number"    "v3"
[6] "var"         "var_name"
> print(ls(pattern = "var"))
[1] "factor_var" "var"        "var_name"
> print(ls(all.name = TRUE))
[1] "factor_var"  ".married"   "my_height"   "my_name"
[5] "my_number"   ".Random.seed" ".rex"       "v3"
[9] "var"         "var_name"
>

```

The RStudio environment pane shows the current workspace.

Figure 18: ls() for hidden variable

- Delete Variable with **rm()**:

- i. Single Variable: Use single variable as parameter to the **rm()** function to delete single variable.

The screenshot shows the RStudio interface with the following console output:

```

> ls()
[1] "a"      "ans"    "b"      "B1"     "B2"     "c"
[7] "cnames" "code"   "d"      "date1"  "e"      "f"
[13] "g"      "h"      "i"      "j"      "k"      "l"
[19] "l1"     "m"      "mymat"  "n"      "n1"     "n2"
[25] "p"      "q"      "rnames"  "s1"     "temp_var" "time1"
[31] "v1"     "v2"     "x"      "y"
> rm(temp_var)
> ls()
[1] "a"      "ans"    "b"      "B1"     "B2"     "c"      "cnames"  "code"
[9] "d"      "date1"  "e"      "f"      "g"      "h"      "i"      "j"
[17] "k"      "l"      "l1"     "m"      "mymat"  "n"      "n1"     "n2"
[25] "p"      "q"      "rnames"  "s1"     "time1"   "v1"     "v2"     "x"
[33] "y"
>

```

The right pane shows the help documentation for the **rm()** function.

Figure 19: rm() function

- ii. Multiple Variable: Use vector having variable as a parameter to **rm()** function to delete them.
- iii. All Variables: Use **list** created by **ls()** function as a parameter to **rm()** function to delete all the variables from workspace except for hidden variables.

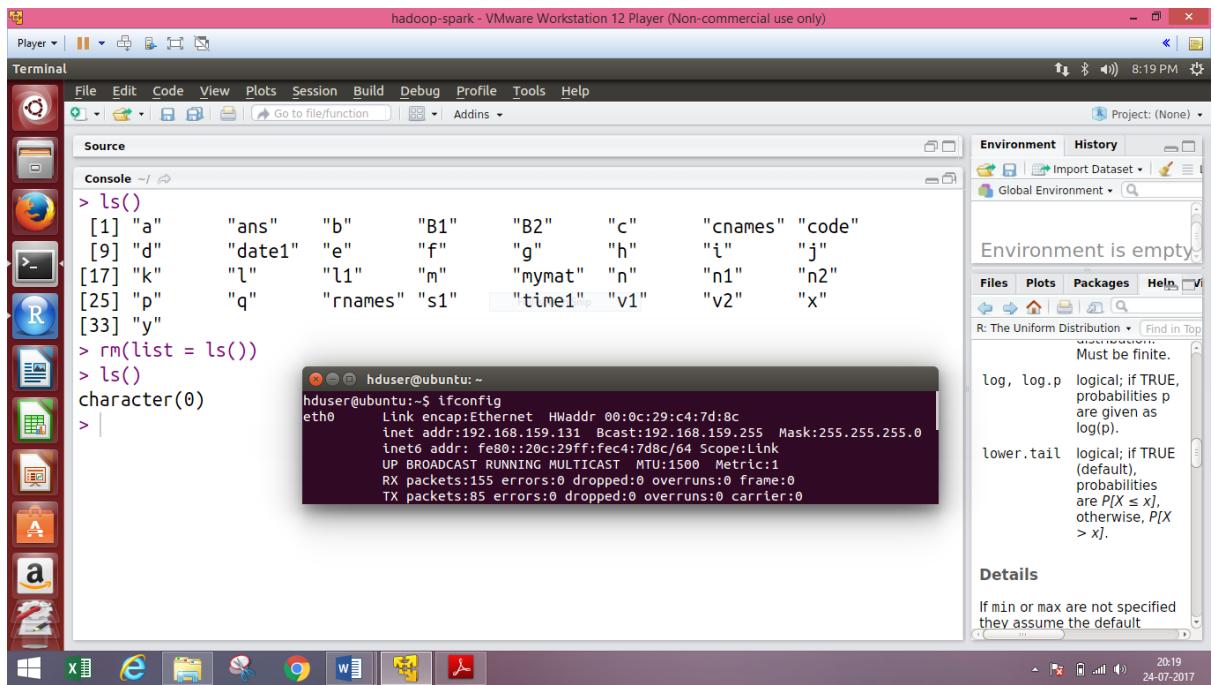


Figure 20: rm all variables in workspace

3. Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

- Arithmetic Operators

+	Adds two vectors
-	Subtracts second vector from the first
*	Multiplies both vectors
/	Divide the first vector with the second
%%	Give the remainder of the first vector with the second
%/%	The result of division of first vector with second (quotient)
^	The first vector raised to the exponent of second vector

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit Code View Plots Session Build Debug Profile Tools Help
Project: (None)
Source
Console ~/ ~
> b <- 1:5
> g <- 11:20
> print(b+g)
[1] 12 14 16 18 20 17 19 21 23 25
> print(b-g)
[1] -10 -10 -10 -10 -15 -15 -15 -15 -15
> print(b*g)
[1] 11 24 39 56 75 16 34 54 76 100
> print(b/g)
[1] 0.09090909 0.16666667 0.23076923 0.28571429 0.33333333
[6] 0.06250000 0.11764706 0.16666667 0.21052632 0.25000000
> print(g%%b)
[1] 0 0 1 2 0 0 1 0 3 0
> print(g%/%b)
[1] 11 6 4 3 3 16 8 6 4 4
> print(b^g)
[1] 1.000000e+00 4.096000e+03 1.594323e+06 2.684355e+08
[5] 3.051758e+10 1.000000e+00 1.310720e+05 3.874205e+08

```

Figure 21: Arithmetic Operators

- Relational Operators

<	Checks if each element of the first vector is greater than the corresponding element of the second vector.
>	Checks if each element of the first vector is less than the corresponding element of the second vector.
==	Checks if each element of the first vector is equal to the corresponding element of the second vector.
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.

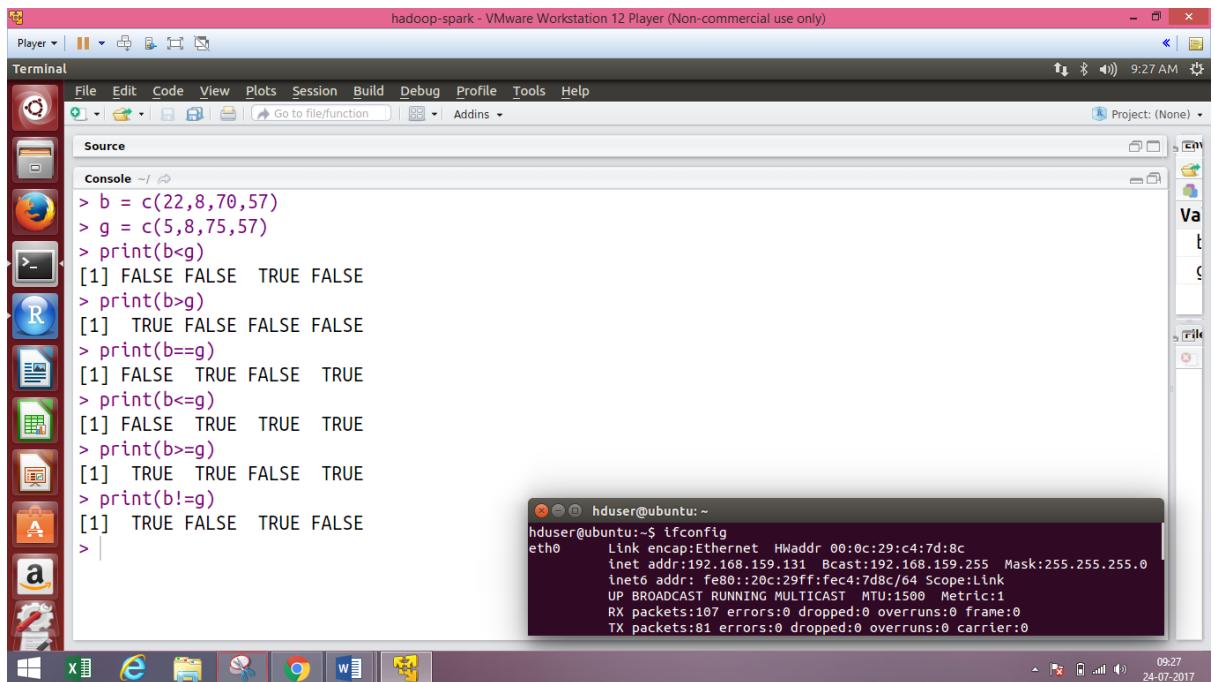


Figure 22: Relational Operators

- Logical Operators

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives the output TRUE if both the elements are TRUE.
 	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.
!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.
&&	It is called as Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE. It considers only the first element of the vectors and give a vector of single element as output.
 	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE. It considers only the first element of the vectors and give a vector of single element as output.

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like a terminal, browser, and file manager. The main window has a 'Terminal' tab selected, showing R code and its output. The R code demonstrates logical operators:

```

> v <- c(3,1,TRUE,2+3i)
> t <- c(4,1,FALSE,2+3i)
> print(v&t)
[1] TRUE TRUE FALSE TRUE
> print(v|t)
[1] TRUE TRUE TRUE TRUE
> print(!v)
[1] FALSE FALSE FALSE FALSE
> v <- c(3,0,F,5+7i)
> print(!v)
[1] FALSE TRUE TRUE FALSE
>

```

Below the terminal, a terminal window titled 'hduser@ubuntu:' shows the command 'ifconfig' and its output, indicating network interface details.

Figure 23: Logical Operators

- Assignment Operators

<- == <--	Left Assignment Operators
-> ->	Right Assignment Operators

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like a terminal, browser, and file manager. The main window has a 'Terminal' tab selected, showing R code and its output. The R code demonstrates assignment operators:

```

> v1 <- c(3,1,TRUE,2+3i)
> v2 <- c(3,1,TRUE,2+3i)
> v3 = c(3,1,TRUE,2+3i)
> print(v1)
[1] 3+0i 1+0i 1+0i 2+3i
> print(v2)
[1] 3+0i 1+0i 1+0i 2+3i
> print(v3)
[1] 3+0i 1+0i 1+0i 2+3i
> c(3,1,TRUE,2+3i) -> v1
> c(3,1,TRUE,2+3i) -> v2
> print(v1)
[1] 3+0i 1+0i 1+0i 2+3i
> print(v2)
[1] 3+0i 1+0i 1+0i 2+3i
>

```

Below the terminal, a terminal window titled 'hduser@ubuntu:' shows the command 'ifconfig' and its output, indicating network interface details.

Figure 24: Assignment Operators

- Miscellaneous Operators

:	It is Colon operator. It creates the series of numbers in sequence for a vector.
----------	--

%in%	This operator is used to identify if an element belongs to a vector.
%*%	This operator is used to multiply a matrix with its transpose.

The screenshot shows the RStudio interface with a terminal window open. The terminal window displays the following R session:

```

> v <- 2:8
> print(v)
[1] 2 3 4 5 6 7 8
> n1 <- 8
> n2 <- 12
> t <- 1:10
> print(n1 %in% t)
[1] TRUE
> print(n2 %in% t)
[1] FALSE
> M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE)
> t = M %*% t(M)
> print(t)
[,1] [,2]
[1,]   65   82
[2,]   82  117
>

```

Below the terminal window, a separate terminal window shows the command `ifconfig` being run on an Ubuntu system, displaying network interface details.

Figure 25: Miscellaneous Operators

4. Functions

- **class** and **mode** of variable/object:
R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Class can be seen as abstract data type here. Mode is the type or storage mode of an object.

Use:

```

class(x)
class(x) <- value
mode(x)
mode(x) <- value

```

The screenshot shows an RStudio interface running on a Windows host. The R console window displays the following R code and its output:

```

> num = 5.7
> cat("num: ", num, " Class: ", class(num), " Mode: ", mode(num))
num: 5.7 Class: numeric Mode: numeric
> inte = 27L
> cat("inte: ", inte, " Class: ", class(inte)," Mode: ", mode(inte))
inte: 27 Class: integer Mode: numeric
> ch = "Gaz"
> cat("ch: ", ch, " Class: ", class(ch)," Mode: ", mode(ch))
ch: Gaz Class: character Mode: character
> r = charToRaw('abc')
> cat("r: ", r, " Class: ", class(r)," Mode: ", mode(r))
r: 61 62 63 Class: raw Mode: raw
> boo = T
> cat("boo: ", boo, " Class: ", class(boo)," Mode: ", mode(boo))
boo: TRUE Class: logical Mode: logical
> comp = 4+9i
> cat("comp: ", comp, " Class: ", class(comp)," Mode: ", mode(comp))
comp: 4+9i Class: complex Mode: complex

```

Figure 26: *class()* and *mode()* functions

- Type Casting:
 - i. **as.integer()** function: This function is used to convert numeric, complex, logical, raw data in integer.

P.S. Converting Character to Integer generate NA which is introduced by coercion.

Use: `as.integer(x, ...)`

The screenshot shows an RStudio interface running on a Windows host. The R console window displays the following R code and its output:

```

> r <- 13.5
> print(class(r))
[1] "numeric"
> print(mode(r))
[1] "numeric"
> s = as.integer(r)
> print(s)
[1] 13
> print(class(s))
[1] "integer"
> print(mode(s))
[1] "numeric"
>

```

Below the console, a terminal window shows the output of the `ifconfig` command:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131  Bcast:192.168.159.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:609 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0

```

Figure 27: *as.integer()* function

- ii. **as.double()** and **as.numeric()** functions: Create, coerce to or test for a double-precision vector.

Use:

as.double(x, ...)
as.single(x, ...)
as.numeric(x, ...)

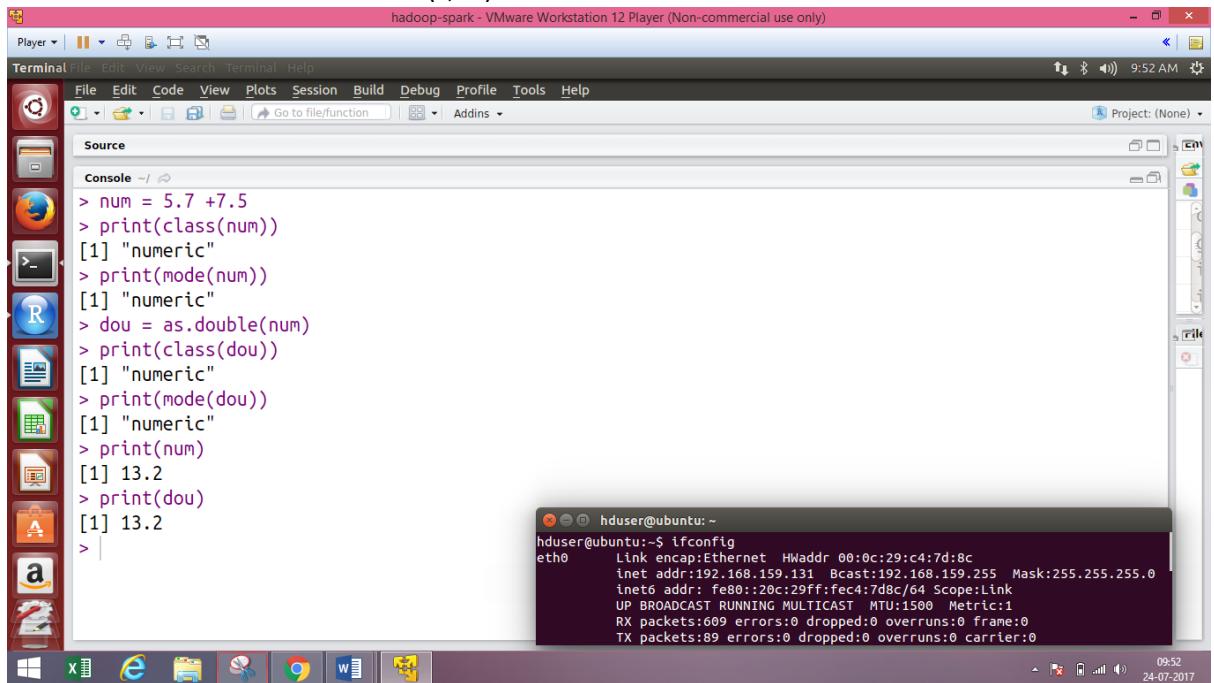


Figure 28: `as.double()` function

iii. **as.character()** function: Create or test for objects of type "character".

Use: `as.character(x, ...)`

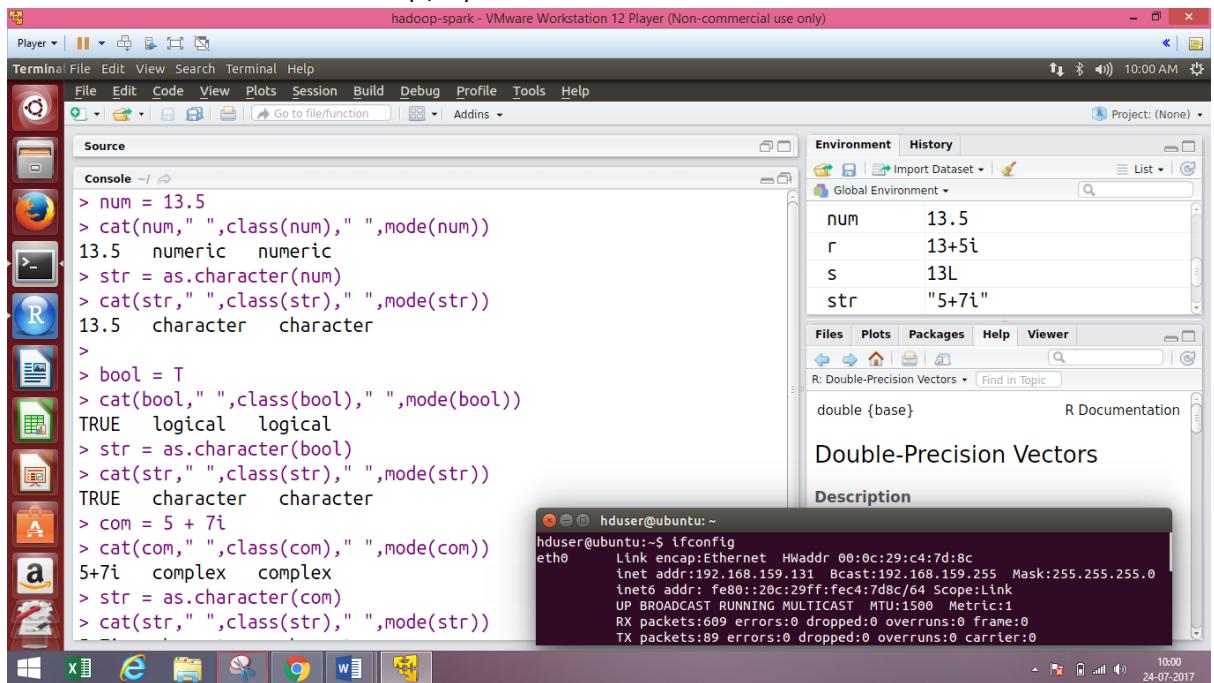


Figure 29: `as.character()` function

- **length()** function:

It gives the number of the objects in the vector. Also see [LIST-LENGTH](#)

Use:

```

length(x)
length(x) <- value

> v1 = c("Gazal", T, 57, 2+3i, 7L, charToRaw('gaz'))
> print(length(v1))
[1] 8
> v1
[1] "Gazal" "TRUE"   "57"     "2+3i"   "7"      "67"     "61"
[8] "7a"
>

```

hduser@ubuntu:~\$ ifconfig
eth0 Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
inet6 addr: fe80::20c:29ff:fe47:d8c/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:609 errors:0 dropped:0 overruns:0 frame:0
TX packets:89 errors:0 dropped:0 overruns:0 carrier:0

Figure 30: length() function

- **nchar()** function:

Gives vector f sizes of the corresponding elements of provided vector.

Use:

```

nchar(x, type = "chars", allowNA = FALSE)
nzchar(x)

```

```

> v1 = c("Gazal", T, 57, 2+3i, 7L, charToRaw('gaz'))
> cat("v1: ", v1, " Length: ", length(v1))
v1: Gazal TRUE 57 2+3i 7 67 61 7a  Length:  8
> cat("nchar: ", nchar(v1))
nchar:  5 4 2 4 1 2 2 2
> v2 = c(NA, "Patel", NA, F)
> v1 = append(v1, v2)
> cat("v1: ", v1, " Length: ", length(v1))
v1: Gazal TRUE 57 2+3i 7 67 61 7a NA Patel NA FALSE  Length:  12
> cat("nchar: ", nchar(v1, type = "chars", allowNA = FALSE ))
nchar:  5 4 2 4 1 2 2 2 2 5 2 5
> cat("nchar: ", nchar(v1, type = "width", allowNA = FALSE ))
nchar:  5 4 2 4 1 2 2 2 2 5 2 5
> cat("nchar: ", nchar(v1, type = "byte", allowNA = FALSE ))
nchar:  5 4 2 4 1 2 2 2 2 5 2 5
> cat("nchar: ", nchar(v1, type = "byte", allowNA = T ))
nchar:  5 4 2 4 1 2 2 2 2 5 2 5
>

```

hduser@ubuntu:~\$ ifconfig
eth0 Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
inet6 addr: fe80::20c:29ff:fe47:d8c/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:609 errors:0 dropped:0 overruns:0 frame:0
TX packets:89 errors:0 dropped:0 overruns:0 carrier:0

Figure 31: nchar() function

- **c()** function: creates vector of any object.

```

> var <- list(c(5,7,TRUE),21.3,sin,"Gazal")
> print(var)
[[1]]
[1] 5 7 1

[[2]]
[1] 21.3

[[3]]
function (x) .Primitive("sin")

[[4]]
[1] "Gazal"

> print(class(var))
[1] "list"
> print(mode(var))
[1] "list"

```

Figure 32: c() function

- **append()** function:

It appends any element (Singular or Atomic variable or function) at the end of the given vector. Value for 'after' can be given optionally which is a subscript, after which the values are to be appended.

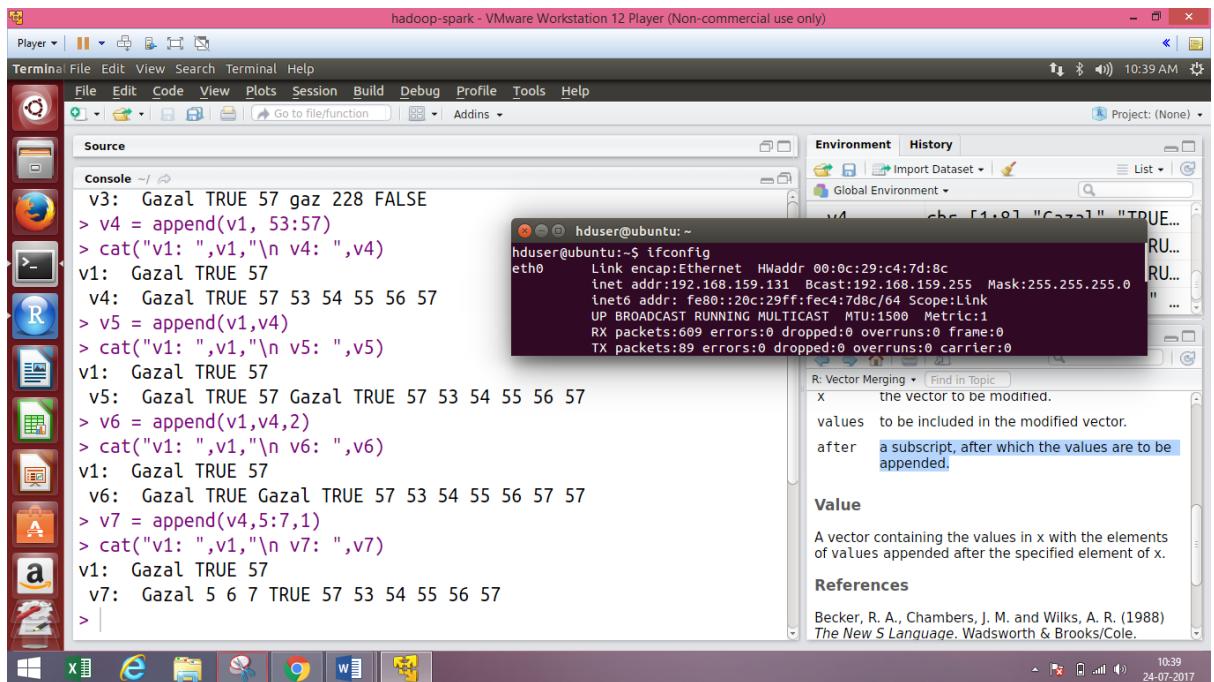
Use: append(x, values, after = length(x))

```

> v1 = c("Gazal", T, 57)
> v2 = append(v1,"Patel")
> cat("v1: ",v1,"\n v2: ",v2)
v1: Gazal TRUE 57
v2: Gazal TRUE 57 Patel
> v3 = append(v1, c("gaz", 228 , F))
> cat("v1: ",v1,"\n v3: ",v3)
v1: Gazal TRUE 57
v3: Gazal TRUE 57 gaz 228 FALSE
> v4 = append(v1, 53:57)
> cat("v1: ",v1,"\n v4: ",v4)
v1: Gazal TRUE 57
v4: Gazal TRUE 57 53 54 55 56 57
> v5 = append(v1,v4)
> cat("v1: ",v1,"\n v5: ",v5)
v1: Gazal TRUE 57
v5: Gazal TRUE 57 Gazal TRUE 57 53 54 55 56 57
>

```

Figure 33: append() function

Figure 34: `append()` function with 'after' parameter

- **seq**

Generate regular sequences.

- seq() or seq.default()** functions: `seq` is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.

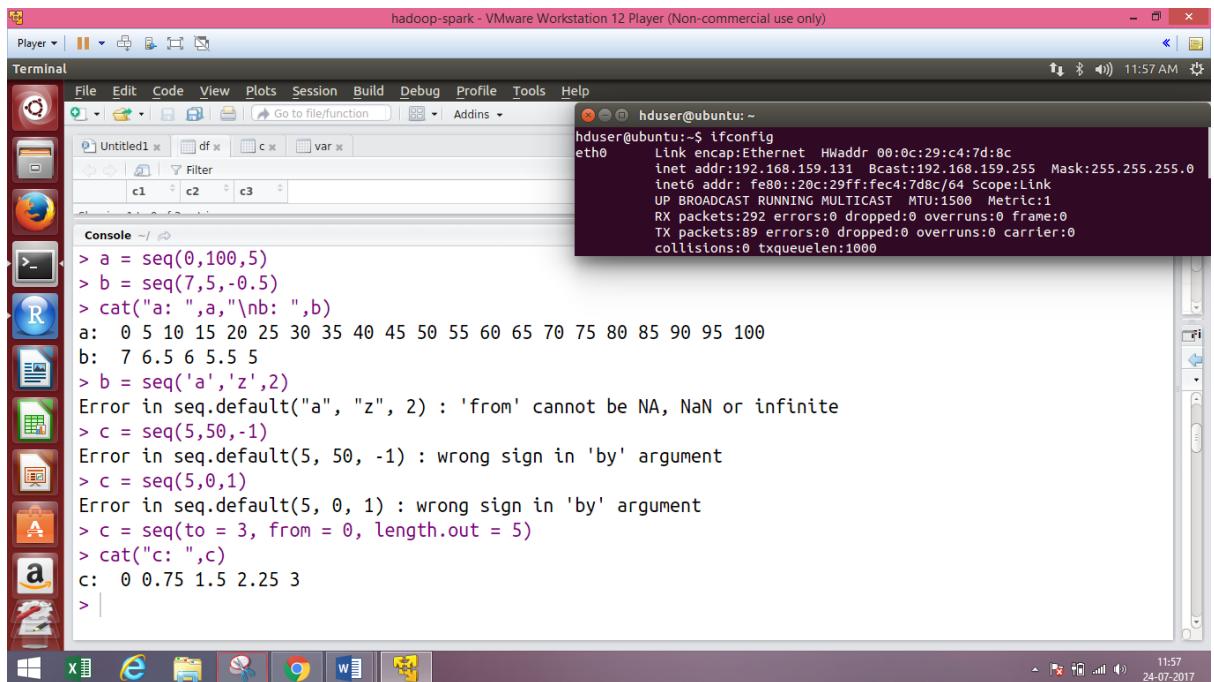
Use:

```

seq(...)
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out =
NULL, along.with = NULL, ...)
seq(from, to)
seq(from, to, by= )
seq(from, to, length.out= )
seq(along.with= )
seq(from)
seq(length.out=)
```

Basic Programming Assignment-2

Gazal Patel - DSFT1701173



The screenshot shows an RStudio interface running on a Windows host. The terminal window displays the following R session:

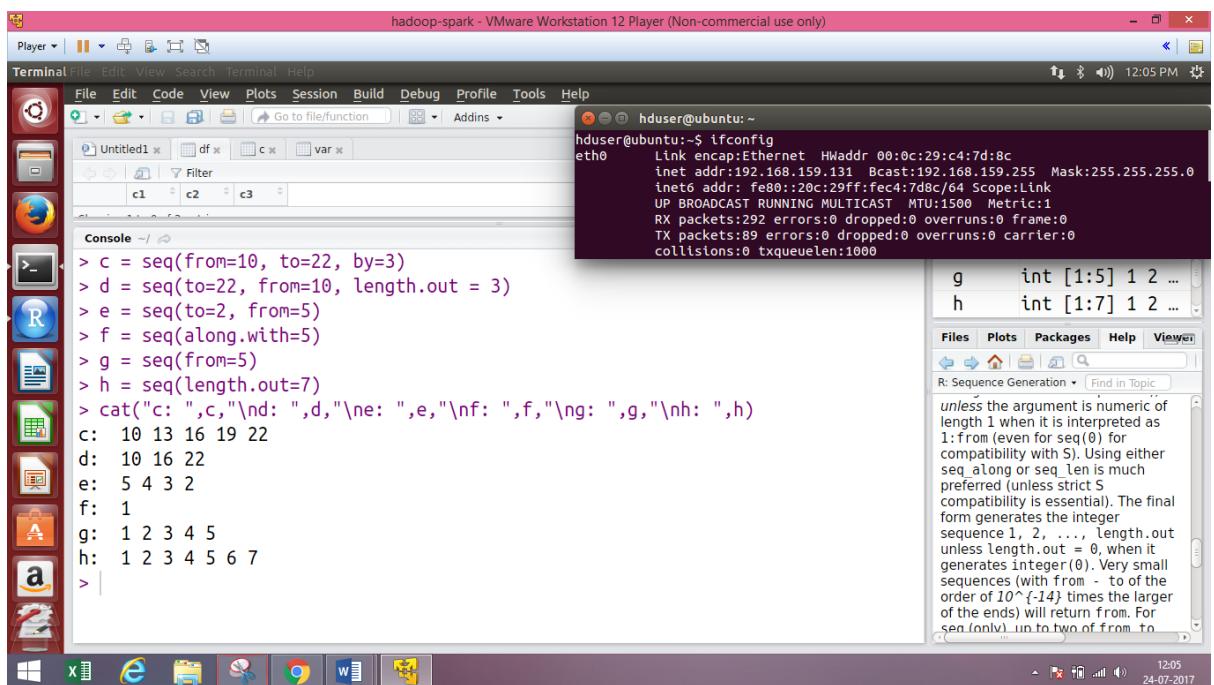
```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:292 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000

> a = seq(0,100,5)
> b = seq(7.5,-0.5)
> cat("a: ",a,"\nb: ",b)
a:  0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
b:  7 6.5 6 5.5 5
> b = seq('a','z',2)
Error in seq.default("a", "z", 2) : 'from' cannot be NA, NaN or infinite
> c = seq(5,50,-1)
Error in seq.default(5, 50, -1) : wrong sign in 'by' argument
> c = seq(5,0,1)
Error in seq.default(5, 0, 1) : wrong sign in 'by' argument
> c = seq(to = 3, from = 0, length.out = 5)
> cat("c: ",c)
c:  0 0.75 1.5 2.25 3
>

```

Figure 35: seq() function



The screenshot shows an RStudio interface running on a Windows host. The terminal window displays the following R session, which includes a help documentation call:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:292 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000

> c = seq(from=10, to=22, by=3)
> d = seq(to=22, from=10, length.out = 3)
> e = seq(to=2, from=5)
> f = seq(along.with=5)
> g = seq(from=5)
> h = seq(length.out=7)
> cat("c: ",c,"\nd: ",d,"\ne: ",e,"\nf: ",f,"g: ",g,"h: ",h)
c:  10 13 16 19 22
d:  10 16 22
e:  5 4 3 2
f:  1
g:  1 2 3 4 5
h:  1 2 3 4 5 6 7
>

```

A tooltip for the `seq` function is visible on the right side of the screen, providing detailed information about its parameters and behavior.

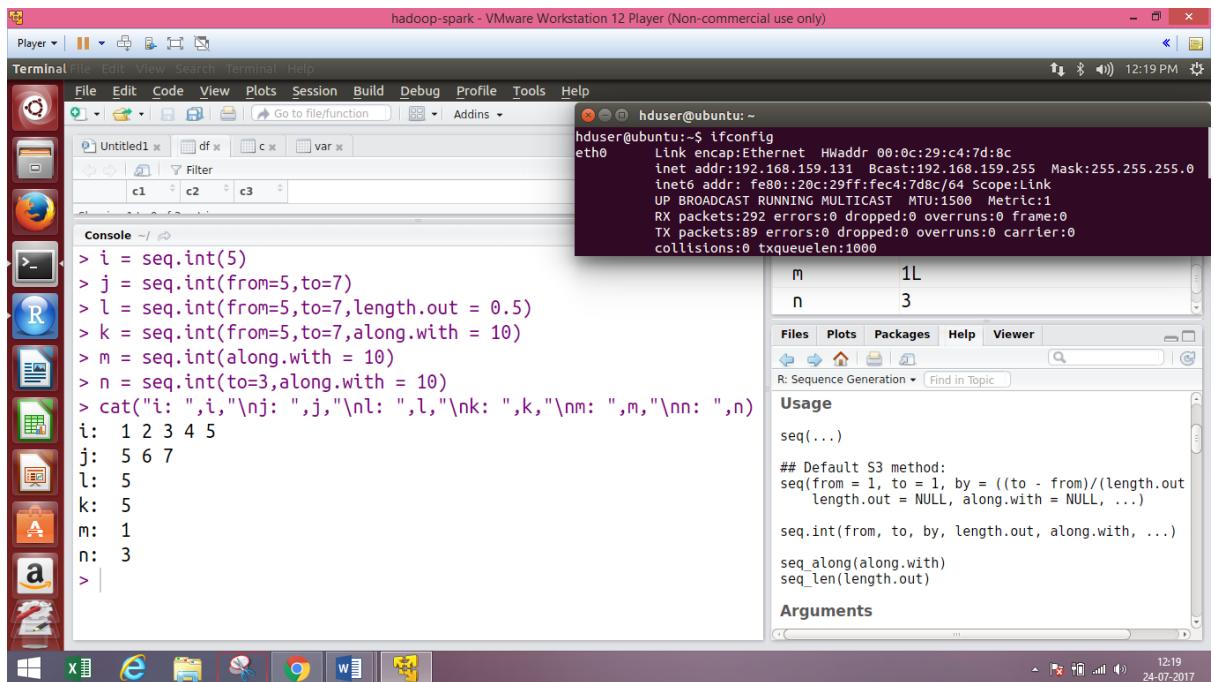
Figure 36: seq() parameter variance

- ii. **seq.int()** function: It generates regular sequences. It is standard generic with default method which is faster but has few restrictions.

Use: `seq.int(from, to, by, length.out, along.with, ...)`

Basic Programming Assignment-2

Gazal Patel - DSFT1701173



The screenshot shows the RStudio interface with a terminal window open. The terminal shows the following R session:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:292 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000

          M       1L
          n       3

Files Plots Packages Help Viewer
R: Sequence Generation Find in Topic

Usage
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out
  length.out = NULL, along.with = NULL, ...))

seq.int(from, to, by, length.out, along.with, ...)
seq_along(along.with)
seq_len(length.out)

Arguments

```

The R console shows the results of the seq.int() function calls:

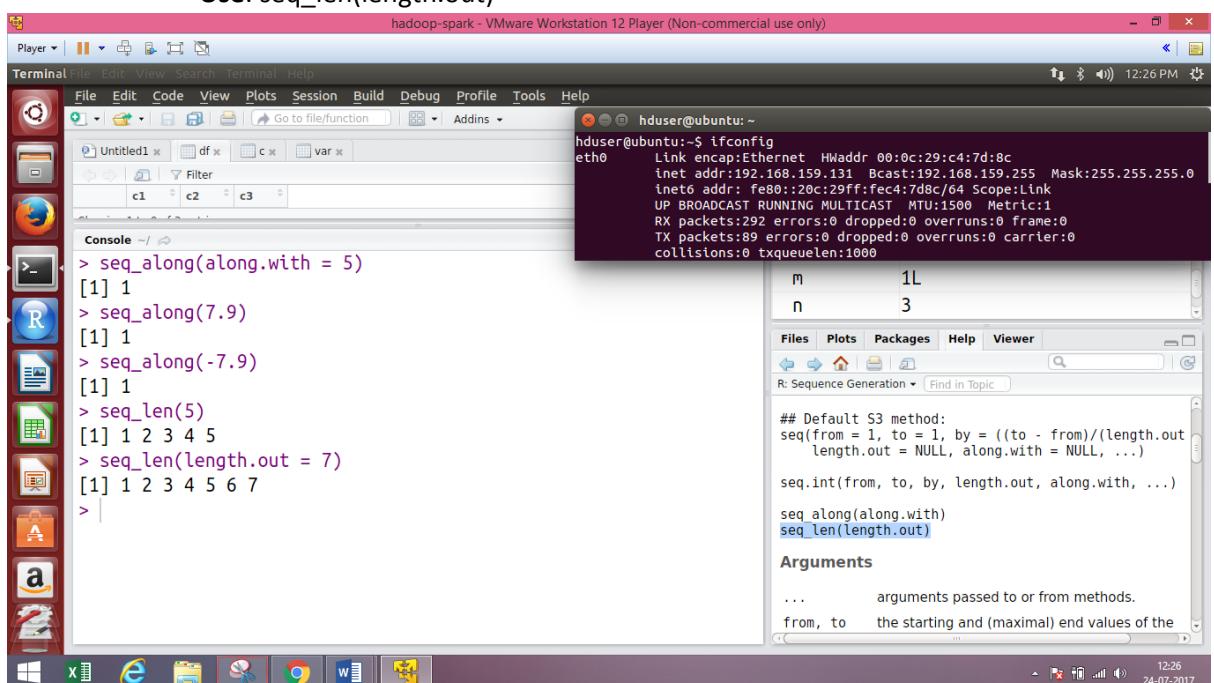
```

> i = seq.int(5)
> j = seq.int(from=5,to=7)
> l = seq.int(from=5,to=7,length.out = 0.5)
> k = seq.int(from=5,to=7,along.with = 10)
> m = seq.int(along.with = 10)
> n = seq.int(to=3,along.with = 10)
> cat("i: ",i,"j: ",j,"l: ",l,"nk: ",k,"nm: ",m,"nn: ",n)
i: 1 2 3 4 5
j: 5 6 7
l: 5
k: 5
m: 1
n: 3
>

```

Figure 37: seq.int() function

- iii. **seq.along()** function: seq is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.
Use: seq_along(along.with)
- iv. **seq_len()** function: seq is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.
Use: seq_len(length.out)



The screenshot shows the RStudio interface with a terminal window open. The terminal shows the following R session:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:292 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000

          M       1L
          n       3

Files Plots Packages Help Viewer
R: Sequence Generation Find in Topic

Usage
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out
  length.out = NULL, along.with = NULL, ...))

seq.int(from, to, by, length.out, along.with, ...)
seq_along(along.with)
seq_len(length.out)

Arguments
...           arguments passed to or from methods.
from, to     the starting and (maximal) end values of the

```

The R console shows the results of the seq_along() and seq_len() function calls:

```

> seq_along(along.with = 5)
[1] 1
> seq_along(7.9)
[1] 1
> seq_along(-7.9)
[1] 1
> seq_len(5)
[1] 1 2 3 4 5
> seq_len(length.out = 7)
[1] 1 2 3 4 5 6 7
>

```

Figure 38: seq_along() and seq_len() functions

- **min(), max(), median(), which.min() and which.max() functions:**

These min(), max() and median() functions determines the minimum and maximum and median of the vector respectively. Which.min() and which.max() functions determines the location of minimum and maximum value in vector respectively, i.e., index of the (first) minimum or maximum of a numeric (or logical) vector.

For a logical vector x, which.min(x) and which.max(x) return the index of the first FALSE or TRUE, respectively.

Use:

```
max(..., na.rm = FALSE)
min(..., na.rm = FALSE)
which.min(x)
which.max(x)
```

P.S. x in above function is either integer or double.

```
> v1
[1] 5 10 75 1 57 22 8 507 10 13 9 17 21
> cat("Maximum: ",max(v1)," Minimum: ",min(v1),"\\nMax at: ",which.max(v1)," Min at: ",which.min(v1))
Maximum: 507 Minimum: 1
Max at: 8 Min at: 4
> median(v1)
[1] 13
> v2
[1] FALSE FALSE TRUE TRUE FALSE TRUE FALSE TRUE TRUE
[10] TRUE
> cat("Maximum: ",max(v2)," Minimum: ",min(v2),"\\nMax at: ",which.max(v2)," Min at: ",which.min(v2))
Maximum: 1 Minimum: 0
Max at: 3 Min at: 1
> median(v2)
[1] 1
>
```

hduser@ubuntu:~\$ ifconfig

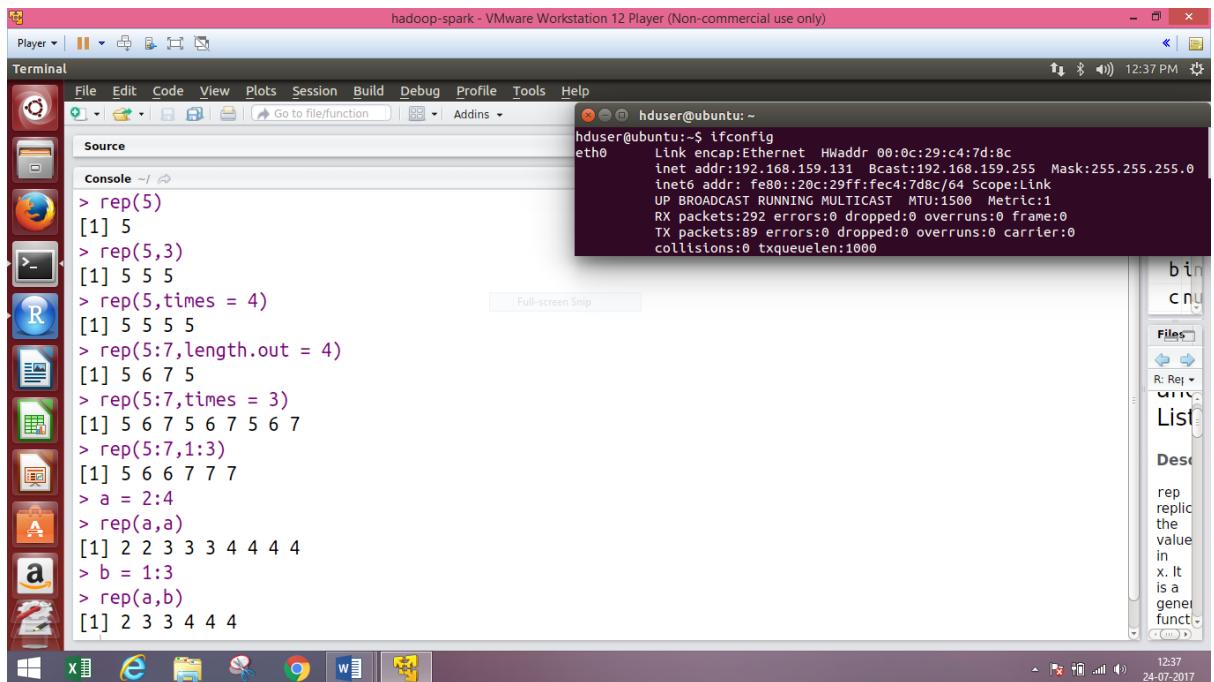
eth0	Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
	inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
	inet6 addr: fe80::20c:29ff:fe47d:8c/64 Scope:Link
	UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
	RX packets:105 errors:0 dropped:0 overruns:0 frame:0

Figure 39: min() and max() functions

- **rep() function:** rep replicates the values in x.

- i. **rep() function:** It is a generic repetition function.

Use: rep(x, ...)



The screenshot shows the RStudio interface with the terminal tab active. The user has run several commands using the `rep()` function:

```

> rep(5)
[1] 5
> rep(5,3)
[1] 5 5 5
> rep(5:7,length.out = 4)
[1] 5 6 7 5
> rep(5:7,times = 3)
[1] 5 6 7 5 6 7 5 6 7
> rep(5:7,1:3)
[1] 5 6 6 7 7 7
> a = 2:4
> rep(a,a)
[1] 2 2 3 3 3 4 4 4 4
> b = 1:3
> rep(a,b)
[1] 2 3 3 4 4 4

```

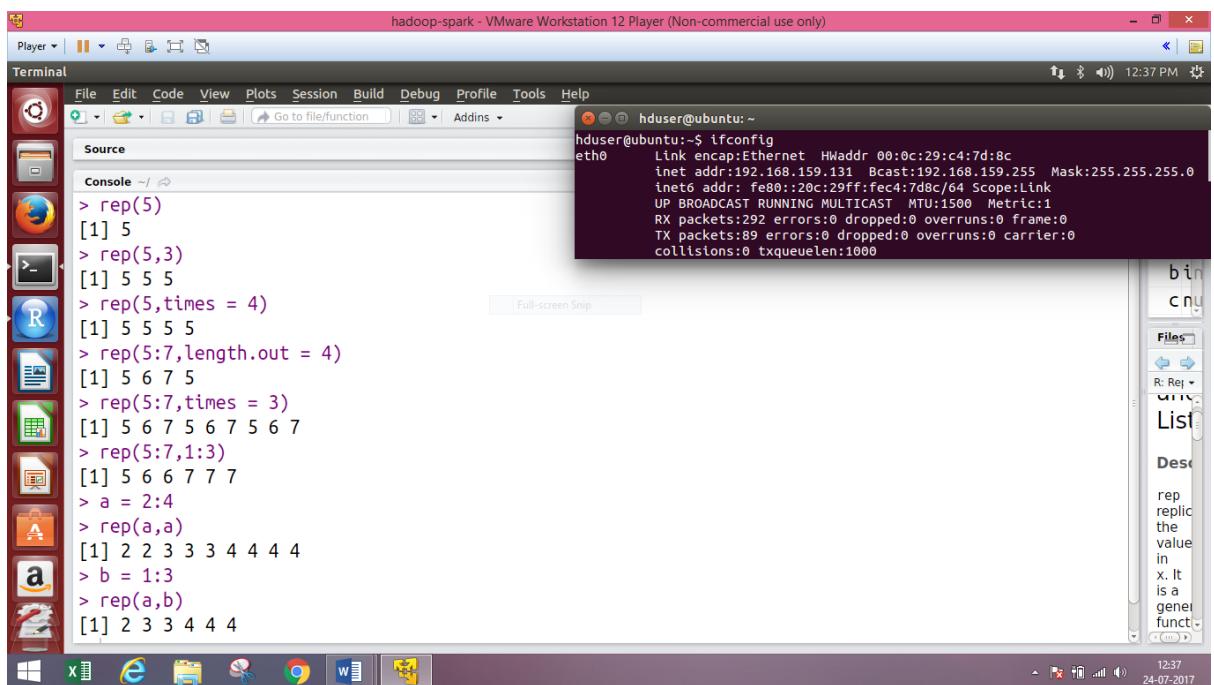
After these commands, the user runs `ifconfig` to check network interface statistics:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:292 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000

```

Figure 40: `rep()` function



This screenshot is identical to Figure 40, showing the RStudio interface with the terminal tab active. The user has run the same sequence of `rep()` function examples and the `ifconfig` command to verify network settings.

Figure 41: `rep()` function

ii. `rep.int()` and `rep_len()` functions:

`rep.int` and `rep_len` are faster simplified versions for two common cases.
They are not generic.

Use:

```

rep.int(x, times)
rep_len(x, length.out)

```

The screenshot shows an RStudio interface running on a Windows host. The terminal window displays the following R session:

```

> a
[1] 2 3 4
> rep.int(5,3)
[1] 5 5 5
> rep.int(2.4,6.9)
[1] 2.4 2.4 2.4 2.4 2.4 2.4
> rep.int(a,a)
[1] 2 2 3 3 3 4 4 4 4
> rep.int(a,times=1:3)
[1] 2 3 3 4 4 4
> rep_len(5,3)
[1] 5 5 5
> rep_len(a,3)
[1] 2 3 4
> rep_len(a,a)
Error in rep_len(a, a) : invalid 'length.out' value
> rep_len(a,7)
[1] 2 3 4 2 3 4 2

```

Simultaneously, a terminal window titled "hduser@ubuntu: ~" is open, showing the output of the ifconfig command:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:105 errors:0 dropped:0 overruns:0 frame:0

```

Figure 42: rep_int() and rep_len() functions

- **runif()** function: It is one of the functions in uniform distribution class from interval min to max. It generates random deviates. It generates random number from 0 to 1. For random variable in other range optional min and max parameters can be used. The characteristics of output from pseudo-random number generators (such as precision and periodicity) vary widely. See [.Random.seed](#)

Use: runif(n, min = 0, max = 1)

The screenshot shows an RStudio interface running on a Windows host. The terminal window displays the following R session:

```

> runif(4)
[1] 0.2732352 0.5728732 0.3793169 0.6818989
> runif(4)
[1] 0.07921678 0.12258531 0.93187755 0.39484767
> runif(4, min=1)
[1] 1 1 1 1
> runif(4, min=0.5)
[1] 0.7933409 0.6489991 0.9673308 0.8960922
> runif(4, max=0.2)
[1] 0.0501516785 0.0856765939 0.0542895968 0.0004938513
> runif(4, max=5)
[1] 3.172515 3.657562 2.234741 2.035043
> runif(4, min=1, max=5)
[1] 2.376509 3.780935 4.995743 2.437235
>

```

The right pane of the RStudio interface shows the documentation for the runif function, including its arguments and usage examples.

Figure 43: runif() function

- **names()** function: It assigns names to the elements of the vector. Here any object can be assigned a name which is the value defined in the vector.

Use:

```
names(x)
names(x) <- value
```

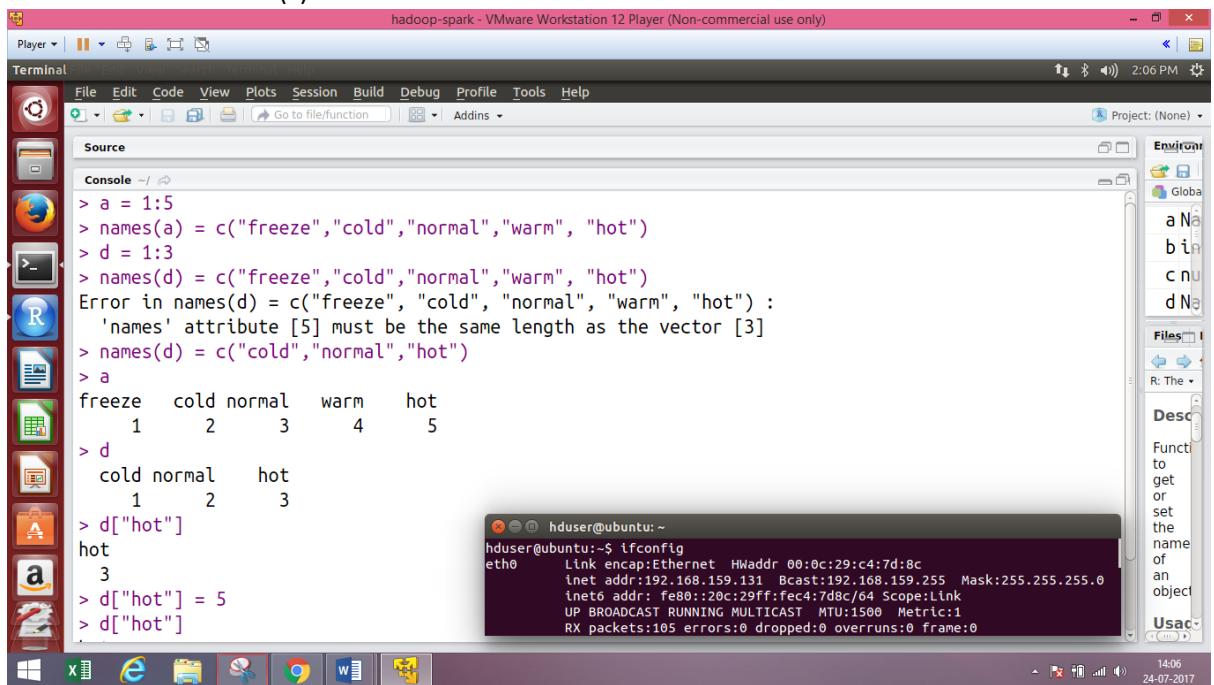


Figure 44: names() function

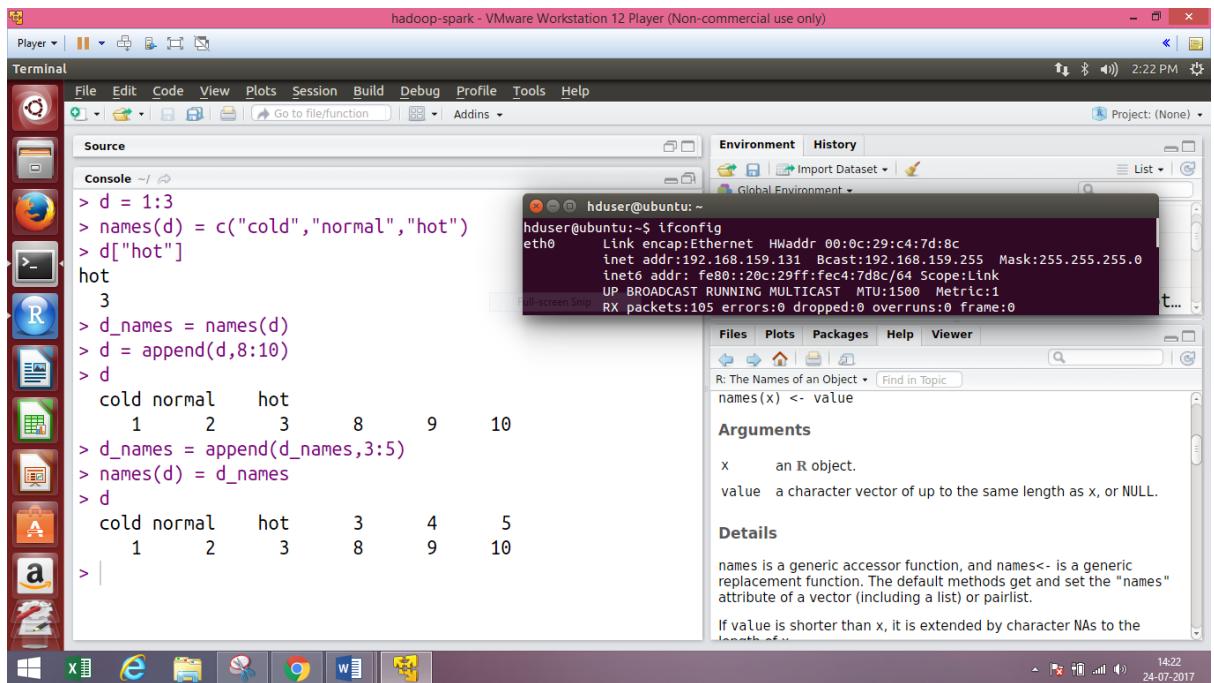


Figure 45: names - use

- **list()** and **pairlist()** functions:

The arguments to list or pairlist are of the form value or tag = value. The functions return a list or dotted pair list composed of its arguments with each value either tagged or untagged, depending on how the argument was specified.

Use:

```
list(...)
```

The screenshot shows the RStudio interface with the following code in the Source pane:

```

pairlist(...)

```

Console output:

```

> l3 = c("Gaz", T, 57)
> list1 = list(l1, l2, l3)
> list1
[[1]]
[1] TRUE FALSE TRUE

[[2]]
[1] 4 5 6 7 8 9

[[3]]
[1] "Gaz"   "TRUE"  "57"

> pairlist(l1, l3)
[[1]]
[1] TRUE FALSE TRUE

[[2]]
[1] "Gaz"   "TRUE"  "57"

```

The Environment pane shows variables l1 and logi defined. The Help pane provides documentation for pairlist.

Figure 46: list() and pairlist() functions

- **Unlist() function:**

Given a list structure x, unlist simplifies it to produce a vector which contains all the atomic components which occur in x.

Use: unlist(x, recursive = TRUE, use.names = TRUE)

The screenshot shows the RStudio interface with the following code in the Source pane:

```

unlist(list1)

```

Console output:

```

> list1
[[1]]
[1] TRUE FALSE TRUE

[[2]]
[1] 4 5 6 7 8 9

[[3]]
[1] "Gaz"   "TRUE"  "57"

> unlist(list1)
[1] "TRUE"   "FALSE"  "TRUE"   "4"      "5"      "6"      "7"      "8"      "9"
[10] "Gaz"    "TRUE"   "57"
> var = unlist(list1)
> var
[1] "TRUE"   "FALSE"  "TRUE"   "4"      "5"      "6"      "7"      "8"      "9"
[10] "Gaz"    "TRUE"   "57"
>

```

The Environment pane shows variables l1 and logi defined. The Help pane provides documentation for unlist.

Figure 47: unlist() function

- **Is() function:** See [LS](#)

- **sort() function:** Sorts the elements in vector.

Use:

sort(x, decreasing = FALSE, ...)

```
sort(x, decreasing = FALSE, na.last = NA, ...)
sort.int(x, partial = NULL, na.last = NA, decreasing = FALSE,
         method = c("shell", "quick"), index.return = FALSE)
```

The screenshot shows the RStudio interface with the following session history:

```

> v1
[1] 5 10 75 1 57 22 8 507 10 13 9 17 21
> sort_v1 = sort(v1)
> sort_v1
[1] 1 5 8 9 10 10 13 17 21 22 57 75 507
> sort_v1 = sort(v1, decreasing = T)
> sort_v1
[1] 507 75 57 22 21 17 13 10 10 9 8 5
[13] 1
> v3 = append(v1, rep(NA,5), after = 5)
> v3
[1] 5 10 75 1 57 NA NA NA NA NA 22 8
[13] 507 10 13 9 17 21
> sort_v3 = sort(v3, decreasing = T)
> sort_v3
[1] 507 75 57 22 21 17 13 10 10 9 8 5
[13] 1
> sort_v3 = sort(v3)

```

The right panel displays the documentation for the `sort` function, specifically the part about the `decreasing` parameter.

Figure 48: `sort()` function and `decreasing` parameter

The screenshot shows the RStudio interface with the following session history:

```

> v3
[1] 5 10 75 1 57 NA NA NA NA NA 22 8
[13] 507 10 13 9 17 21
> sort_v3 = sort(v3, decreasing = T)
> sort_v3
[1] 507 75 57 22 21 17 13 10 10 9 8 5
[13] 1
> sort_v3 = sort(v3)
> sort_v3
[1] 1 5 8 9 10 10 13 17 21 22 57 75
[13] 507
> sort_v3 = sort(v3, decreasing = T, na.last = T)
> sort_v3
[1] 507 75 57 22 21 17 13 10 10 9 8 5
[13] 1 NA NA NA NA NA
> sort_v3 = sort(v3, na.last = T)
> sort_v3
[1] 1 5 8 9 10 10 13 17 21 22 57 75 507 NA NA NA NA NA

```

The right panel displays the documentation for the `sort` function, specifically the part about the `decreasing` and `na.last` parameters.

Figure 49: `sort()` function with `decreasing` and `NA` parameters

- **`rev()` function:** It provides a reversed version of its argument.

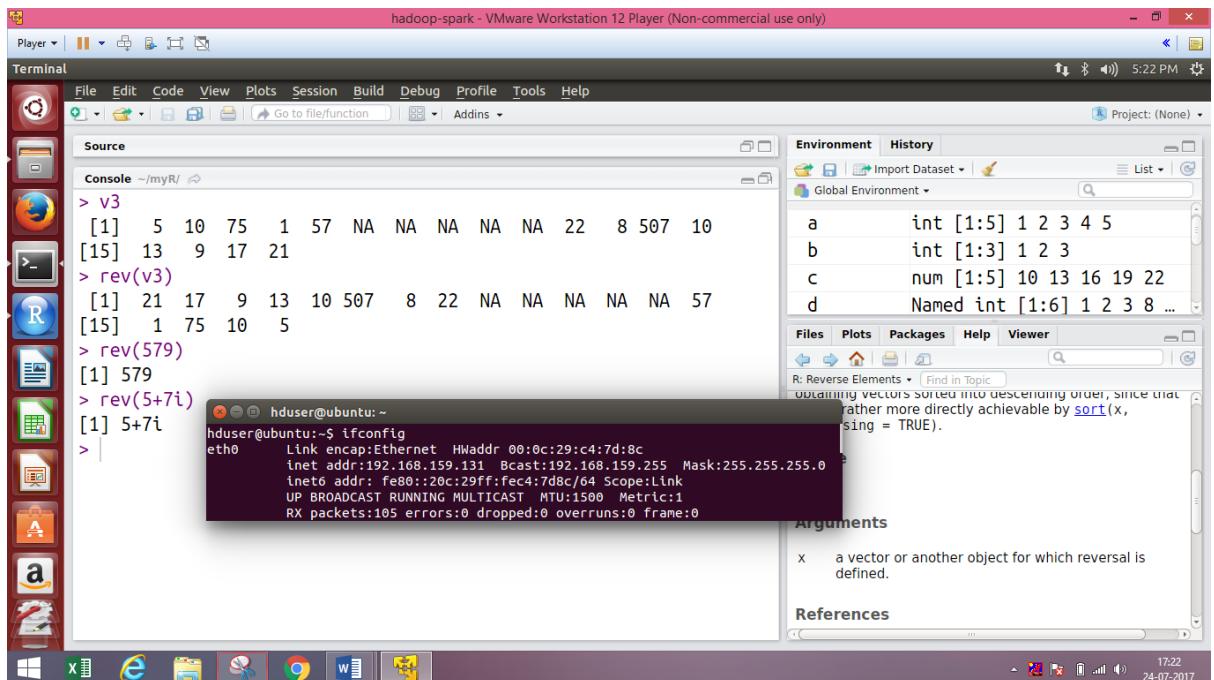


Figure 50: rev() function

- **order()** function:

It returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. sort.list is the same, using only one argument. It returns the position of the value in return.

Use:

```
order(..., na.last = TRUE, decreasing = FALSE)
```

```
sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE,
method = c("shell", "quick", "radix"))
```

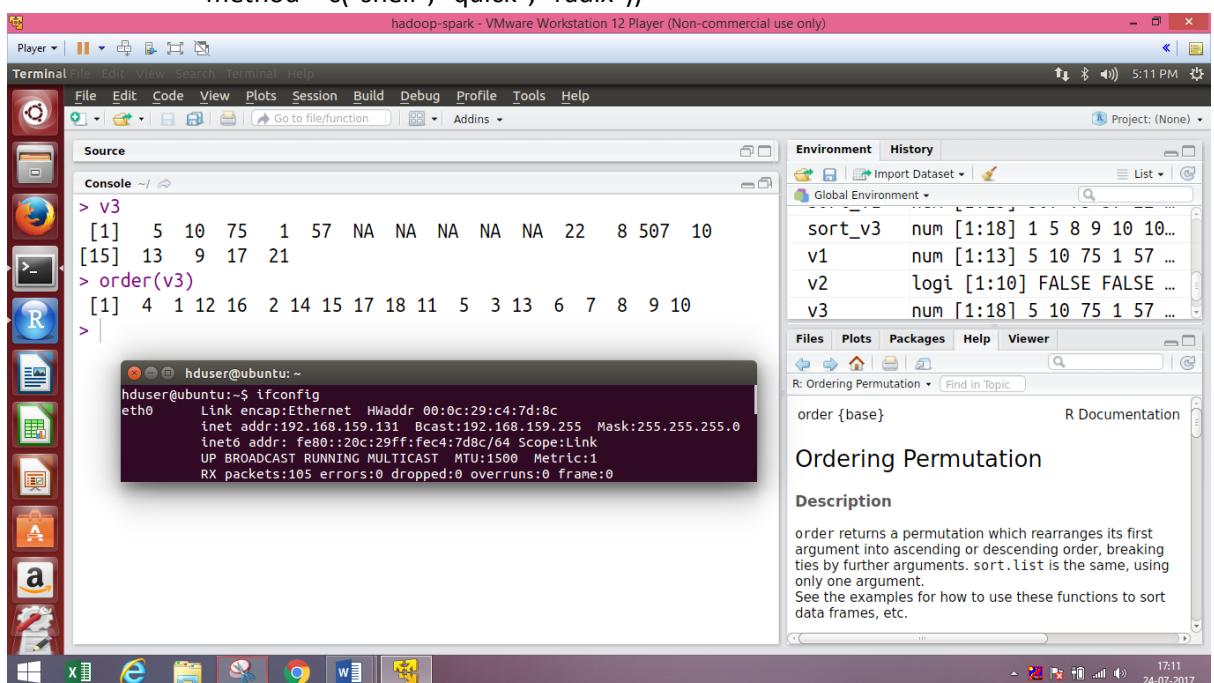


Figure 51: order() function

- **getwd()** function: getwd() returns an absolute filepath representing the current working directory of the R process. getwd returns a character string or NULL if the working directory is not available. On Windows the path returned will use / as the path separator and be encoded in UTF-8. The path will not have a trailing / unless it is the root directory (of a drive or share on Windows).

Use: getwd()

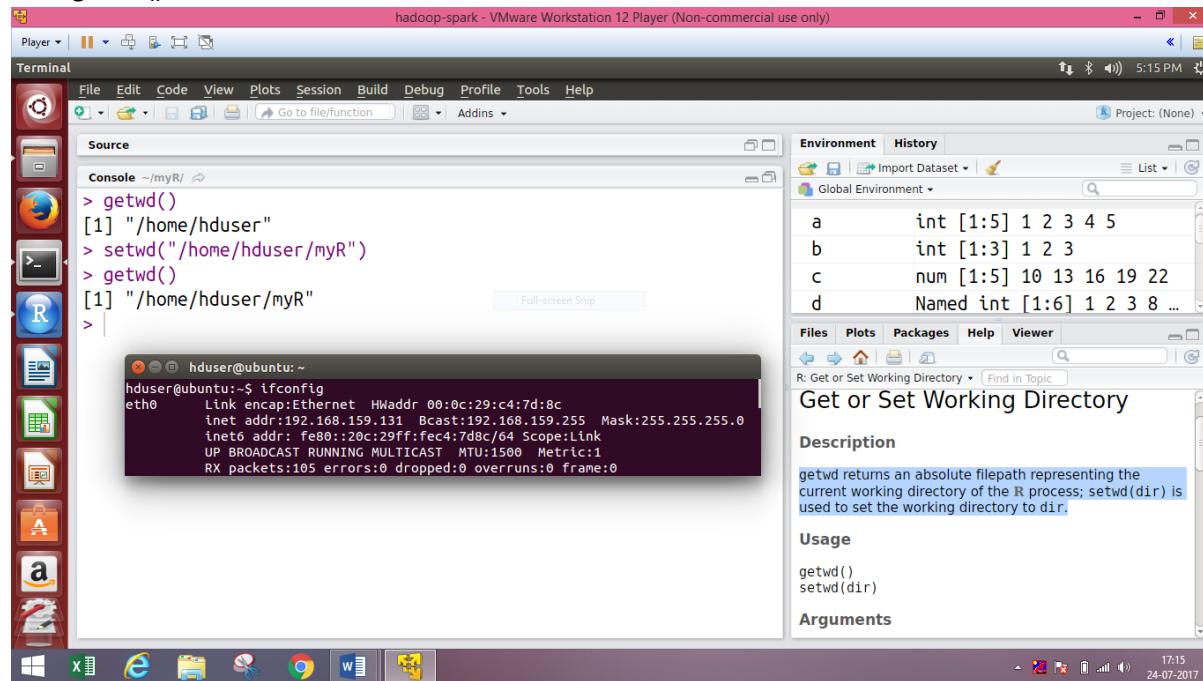


Figure 52: getwd() and setwd() functions

- **setwd()** function: setwd(dir) is used to set the working directory to dir. setwd returns the current directory before the change, invisibly and with the same conventions as getwd. It will give an error if it does not succeed (including if it is not implemented).
- **Use:** setwd(dir)
- **is.element()** function or %in% operator: Finds out if the particular element is in vector or not and returns logical vector.

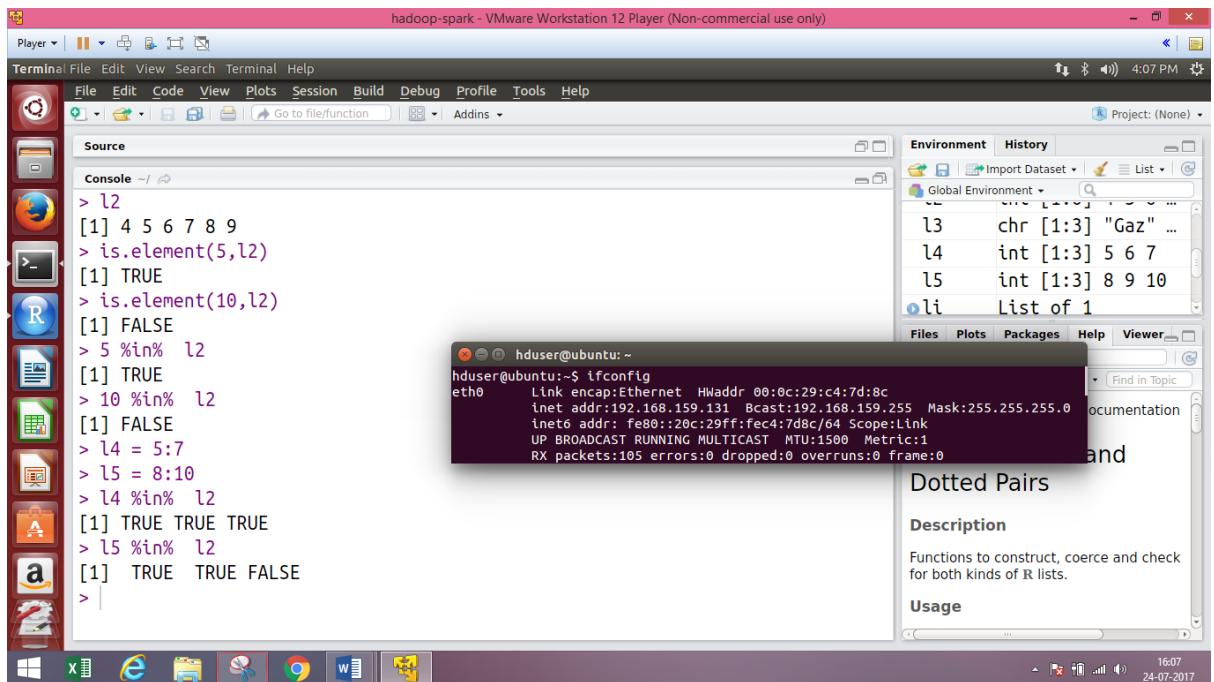


Figure 53: `is.element()` function and `%in%` operator

- **`sample()` function:** sample takes a sample of the specified size from the elements of `x` using either with or without replacement.

Use:

```
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL)
```

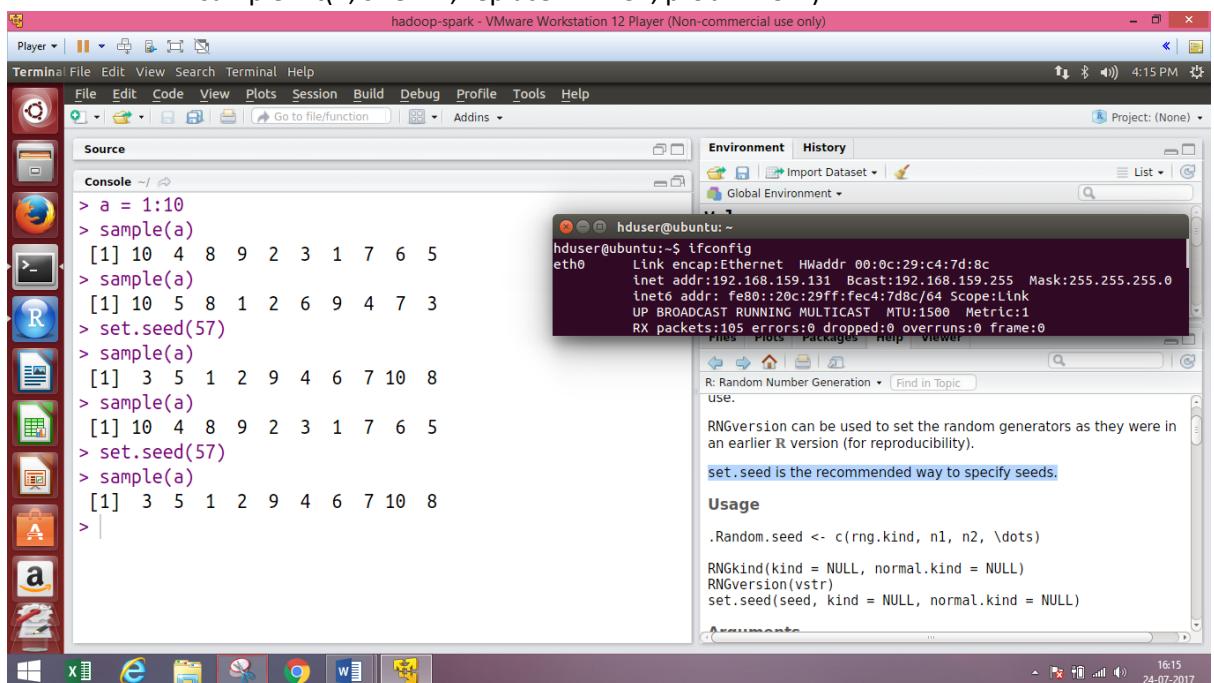


Figure 54: `sample()` and `set.seed()` function

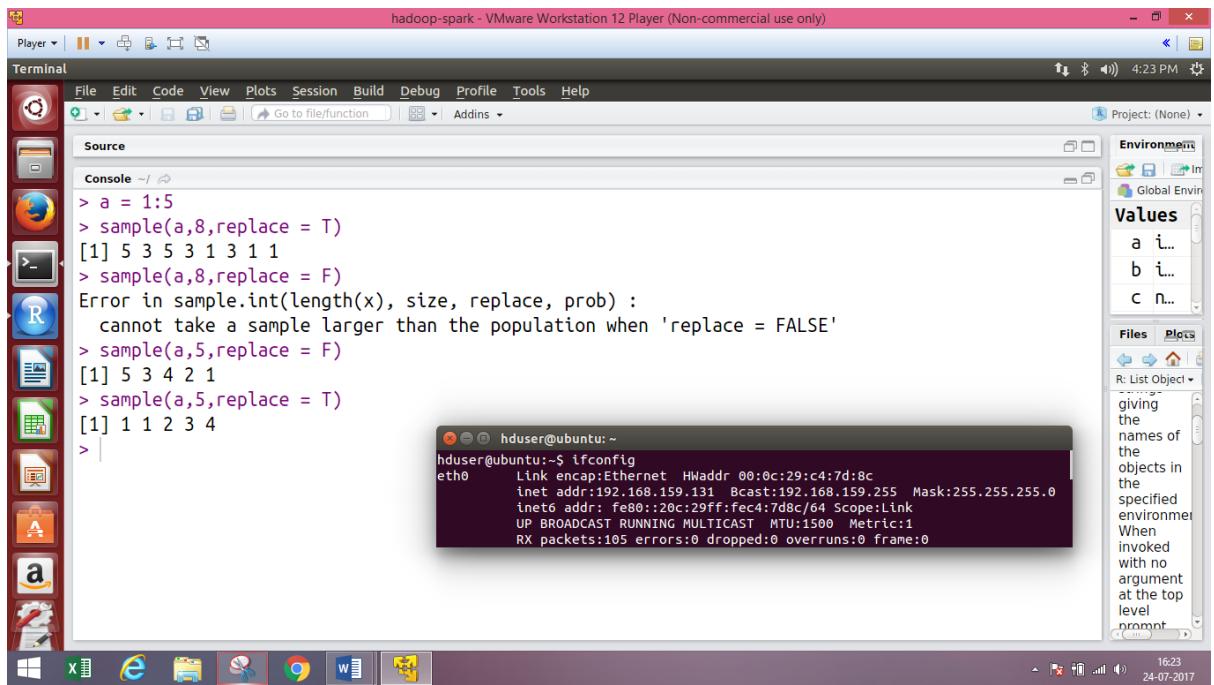


Figure 55: *sample()* with *replace* parameter

- **seed()** function: It is used to change the hidden seed variable which is used to create random variable. In this way same random number can be generated with same seed value arrangements.

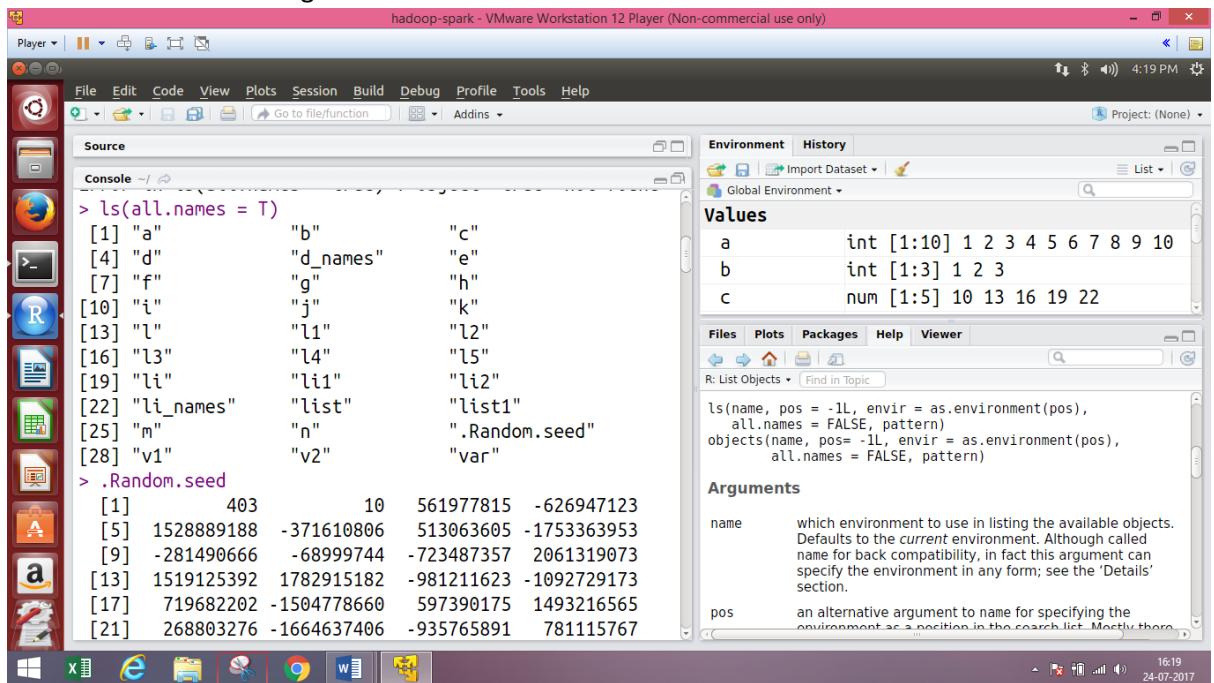


Figure 56: See Hidden Seed Value Arrangement.

- **Sys.Time() and Sys.Date() function:**

Sys.time and Sys.Date returns the system's idea of the current date with and without time.

Sys.time returns an absolute date-time value which can be converted to various time zones and may return different days.

Sys.Date returns the current day in the current timezone.

Use:

```
Sys.time()
Sys.Date()
```

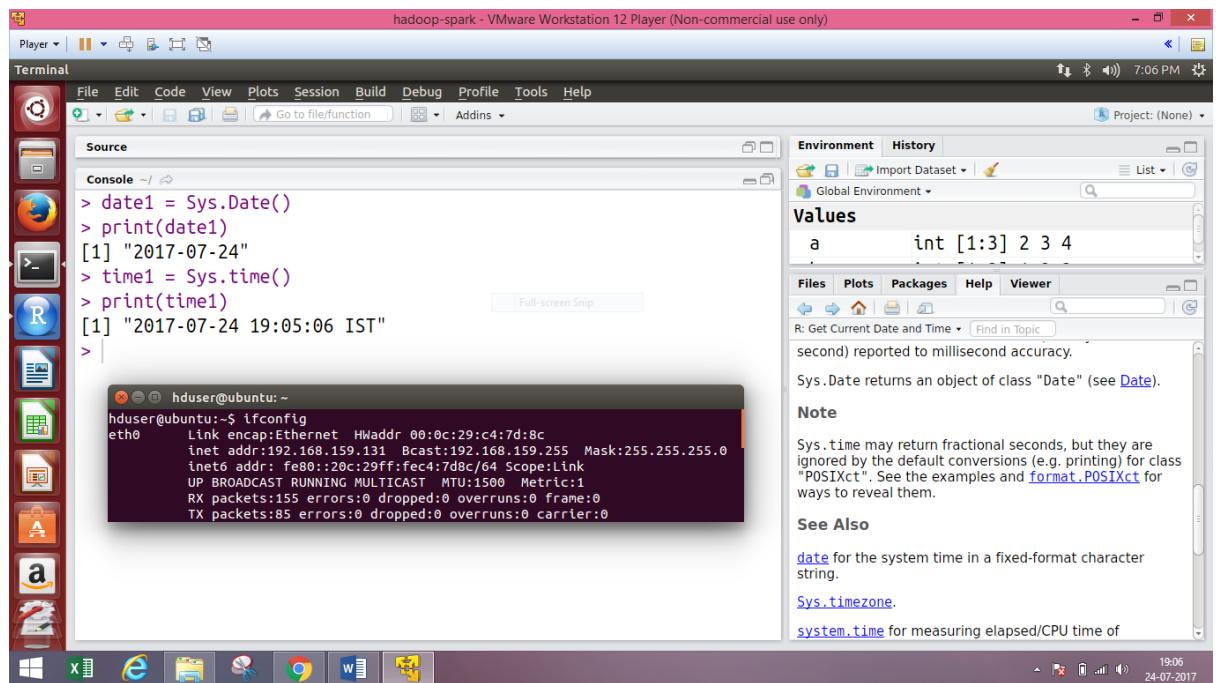


Figure 57: Get System Date and Time

- **Set Operations:**

Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.

Each of **union()**, **intersect()**, **setdiff()** and **setequal()** functions will discard any duplicated values in the arguments, and they apply **as.vector()** function to their arguments (and so in particular coerce factors to character vectors).

is.element(x, y) is identical to **x %in% y**.

Use:

```
union(x, y)
intersect(x, y)
setdiff(x, y)
setequal(x, y)
is.element(el, set)
```

The screenshot shows the RStudio interface running on a Windows host. The terminal window displays R code and its output:

```

> x <- c(sort(sample(1:20, 9)), NA)
> y <- c(sort(sample(3:23, 7)), NA)
> x
[1]  2  6  7  8 11 12 14 16 20 NA
> y
[1]  3  7  9 10 18 20 21 NA
> union(x, y)
[1]  2  6  7  8 11 12 14 16 20 NA  3  9 10 18 21
> intersect(x, y)
[1] 7 20 NA
> setdiff(x, y)
[1]  2  6  8 11 12 14 16
> setdiff(y, x)
[1]  3  9 10 18 21
> setequal(x, y)
[1] FALSE
> setequal(union(x, y),c(setdiff(x, y), intersect(x, y), setdiff(y, x)))
[1] TRUE

```

Below the terminal, the R documentation for the `sets` package is visible, specifically the `Set Operations` section.

Figure 58: set operations

The screenshot shows the RStudio interface running on a Windows host. The terminal window displays R code and its output:

```

> union(x, y)
[1]  2  6  7  8 11 12 14 16 20 NA  3
> intersect(x, y)
[1] 7 20 NA
> setdiff(x, y)
[1]  2  6  8 11 12 14 16
> setdiff(y, x)
[1]  3  9 10 18 21
> setequal(x, y)
[1] FALSE
> setequal(union(x, y),c(setdiff(x, y), intersect(x, y), setdiff(y, x)))
[1] TRUE
> is.element(x, y)
[1] FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE
[10] TRUE
> is.element(y, x)
[1] FALSE TRUE FALSE FALSE FALSE TRUE FALSE TRUE
>

```

Below the terminal, the R documentation for the `sets` package is visible, specifically the `Set Operations` section.

Figure 59: set operations

- **any()** function: This function will check for condition provided as parameter and returns TRUE or FALSE accordingly.

The screenshot shows a Windows desktop environment with a VMware Workstation window titled "hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)". Inside the window, an RStudio interface is displayed. The left sidebar shows icons for various applications like R, RStudio, and a file browser. The main pane has a "Source" tab open, showing R code and its output. The code includes:

```

[1] TRUE
Warning message:
In x > y : longer object length is not a multiple of shorter object length
> p = c(57,50,105,5)
> q = c(75,7,17,21,13,9,4,22)
> any(p>q)
[1] TRUE
> any(q>p)
[1] TRUE
> any(q%p=4)
Error: unexpected input in "any(q%p=4)"
> any(q%p==4)
Error: unexpected input in "any(q%p==4)"
> any(q%p==4)
[1] TRUE
> any(q%/%p==10)
[1] FALSE
>

```

To the right of the code, there is a terminal window showing the command "ifconfig" and its output for interface eth0. Below the terminal, there is a tooltip for the "any" function:

Description
Performs `set union`, intersection, (asymmetric!) difference, equality and membership on two vectors.

Usage
`union(x, y)`
`intersect(x, y)`
`setdiff(x, y)`

The status bar at the bottom right indicates the time as 19:22 and the date as 24-07-2017.

Figure 60: `any()` function

5. Working Directory:

Working directory value is said to be an absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the same directory (for example if symbolic links have been traversed). Also [GETWD](#) and [SETWD](#).

6. Decision Making Conditions:

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **TRUE/T**, and optionally, other statements to be executed if the condition is determined to be **FALSE/F**.

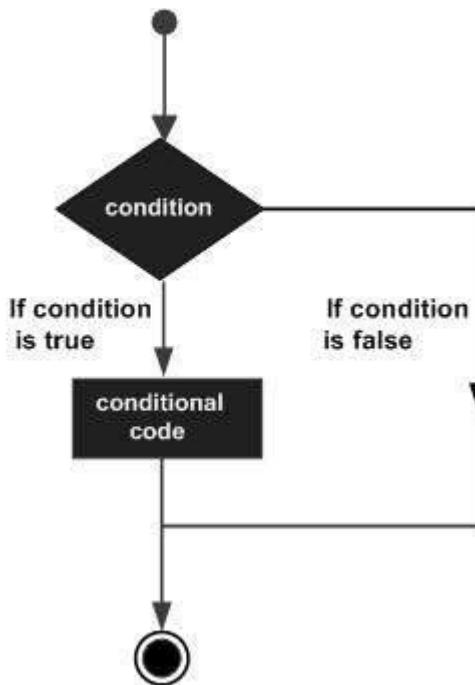


Figure 61: Condition Flow chart

▪ IF Statement

In this condition type an if statement consists of a Boolean expression followed by one or more statements.

A screenshot of the RStudio interface. The left sidebar shows various icons for file operations, while the main workspace contains an R script named 'Untitled1.R'. The script contains the following code:

```

1 a = 18
2 if(a %% 2 == 0 & a %% 3 == 0)
3 {
4   print("a is divisible by 6")
5 }
  
```

Below the script, a 'Console' window shows the output of running the code. The terminal window displays the command 'ifconfig' and its output for the eth0 interface. A tooltip for the 'if' keyword provides a brief explanation of its usage.

Figure 62: If condition

▪ IF...ELSE Statement

In this condition type an if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

The screenshot shows the RStudio interface. In the code editor, there is a script named 'Untitled1.R' containing the following R code:

```

1 a = 19
2 if(a % 2 == 0 & a % 3 == 0)
3 {
4   print("a is divisible by 6")
5 } else
6 {
7   print("a is not divisible by 6")
8 }

```

In the console, the output is:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0

```

The 'Values' pane shows 'a' is 19. A tooltip for 'if' is visible, explaining its behavior with TRUE/FALSE values.

Figure 63: If - Else condition

- **IF...ELSEIF(n)...ELSE Statement**

In this condition type an if statement can be followed by any number of else if statement and then an optional else statement, which executes when the Boolean expression is false.

The screenshot shows the RStudio interface. In the code editor, there is a script named 'Untitled1.R' containing the following R code:

```

1 a = 40
2 if(a % 2 == 0 & a % 3 == 0)
3 {
4   print("a is divisible by 6")
5 } else if( a % 2 == 0 & a % 5 == 0 )
6 {
7   print("a is even number and ends with 0")
8 } else if(a % 2 != 0 & a % 5 == 0 )
9 {
10  print("a is odd ending with 5")
11 } else
12 {
13  print("a is not divisible by 6")
14 }

```

In the console, the output is:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0

```

The 'Values' pane shows 'a' is 19. A tooltip for 'if' is visible, explaining its behavior with TRUE/FALSE values.

Figure 64: If-ElseIf-Else condition

- **Nested If Condition:** If condition block is applied inside other if condition.

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorer, and browser. The main workspace has tabs for 'matrix.R', 'if conditions.R', and 'nestedif.R'. The 'nestedif.R' tab is active, displaying the following R code:

```

> a = 84
> if(a %% 2 == 0 & a %% 3 == 0)
+ {
+   print("a is divisible by 6")
+   if(a %% 7 == 0)
+   {
+     print("a is divisible by 42")
+   }
+ } else
+ {
+   print("a is not divisible by 6")
+ }
[1] "a is divisible by 6"
[1] "a is divisible by 42"
>

```

The 'Console' panel shows the output of the code execution. The 'Environment' panel on the right shows 'a' is assigned the value 84. The 'Plots' and 'Packages' panels are empty. A terminal window at the bottom shows the command 'ifconfig' output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13200 (12.9 KiB) TX bytes:14400 (14.0 KiB)
          Interrupt:15 Memory:f0000000-f0000fff


```

Figure 65: nested if (1/2)

This screenshot is identical to Figure 65, showing the RStudio interface with the same code in 'nestedif.R', the terminal output of 'ifconfig', and the variable 'a' set to 84 in the Environment panel.

Figure 66: nested if (2/2)

▪ Switch Case

A switch statement allows a variable to be tested for equality against a list of values.

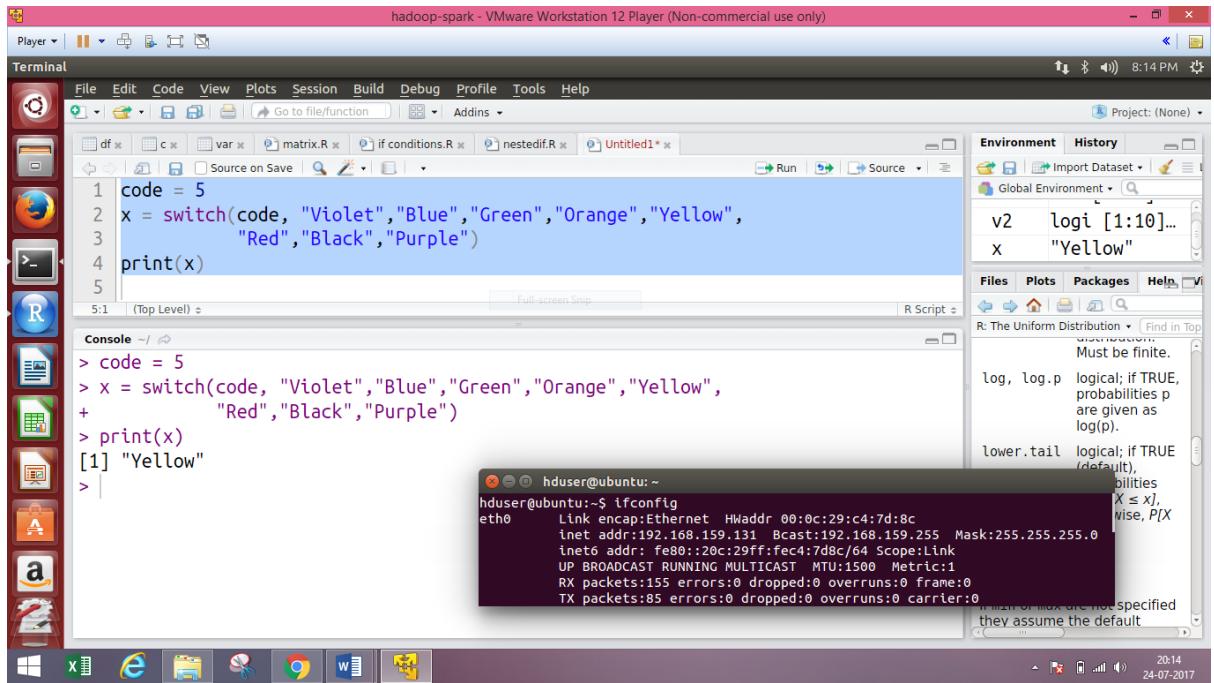


Figure 67: Switch Case

7. Uniform Distribution System

- **`r`unif(), `d`unif, `p`unif, `q`unif:** see [runif\(\) function](#).

`dunif` gives the density, `punif` gives the distribution function, `qunif` gives the quantile function, and `runif` generates random deviates.

The length of the result is determined by `n` for `runif`, and is the maximum of the lengths of the numerical parameters for the other functions.

The numerical parameters other than `n` are recycled to the length of the result. Only the first elements of the logical parameters are used.

Use:

```

dunif(x, min = 0, max = 1, log = FALSE)
punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
runif(n, min = 0, max = 1)

```

8. .Random.seed:

`.Random.seed` is an integer vector, containing the random number generator (RNG) state for random number generation in R. It can be saved and restored, but should not be altered by the user. To manipulate this seed, `set.seed()` function is used where seed is a single value, interpreted as an integer, or `NULL`. See `seed`.

Use:

```

.Random.seed <- c(rng.kind, n1, n2, \dots)
set.seed(seed, kind = NULL, normal.kind = NULL)

```

9. Working with lists:

- **Create list:** See [LIST](#)
- **Access list:**

Basic Programming Assignment-2

Gazal Patel - DSFT1701173

The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R console session:

```
> list1
[[1]]
[1] TRUE FALSE TRUE

[[2]]
[1] 4 5 6 7 8 9

[[3]]
[1] "Gaz" "TRUE" "57"

> list1[[3]]
[1] "Gaz" "TRUE" "57"
> list1[[3]][1]
[1] "Gaz"
> list1[3]
[[1]]
[1] "Gaz" "TRUE" "57"
```

The right pane shows the Global Environment, which contains objects l1, l2, and l3. A terminal window is open in the background showing the output of the ifconfig command.

Figure 68: Access List Elements

The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R console session:

```
> list1[[4]] = c(T,75:79)
> var = list1[[2]]
> list1
[[1]]
[1] TRUE FALSE TRUE

[[2]]
[1] 4 5 6 7 8 9

[[3]]
[1] "Gaz" "TRUE" "57"

[[4]]
[1] 1 75 76 77 78 79

> length(list1)
[1] 4
Loading required namespace: .GlobalEnv
```

The right pane shows the Global Environment, which contains objects n, v1, v2, and list. A terminal window is open in the background showing the output of the ifconfig command. A tooltip for the 'list' function is visible, providing documentation for generic and dotted pairs.

Figure 69: Access List Elements

Basic Programming Assignment-2

Gazal Patel - DSFT1701173

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorers, and browsers. The main console window displays R code and its output:

```
[1] 4 5 6 7 8 9  
$`A Character Vector`  
[1] "Gaz" "TRUE" "57"  
> li<-["A Numeric Vector"]  
$`A Numeric Vector`  
[1] 4 5 6 7 8 9  
  
> li$"A Numeric Vector"[2]  
[1] 5  
> li["A Numeric Vector"][2]  
$<NA>  
NULL  
  
> li[[["A Numeric Vector"]][2]]  
[1] 5  
>
```

Below the console, a terminal window shows the command `ifconfig` output:

```
hduser@ubuntu:~$ ifconfig  
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c  
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0  
          inet6 addr: fe80::20c:29ff:fe47:7d8c/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:105 errors:0 dropped:0 overruns:0 frame:0
```

The right pane of RStudio shows the help documentation for the `list` function, specifically the section on "Dotted Pairs".

Figure 70: Access List with Names

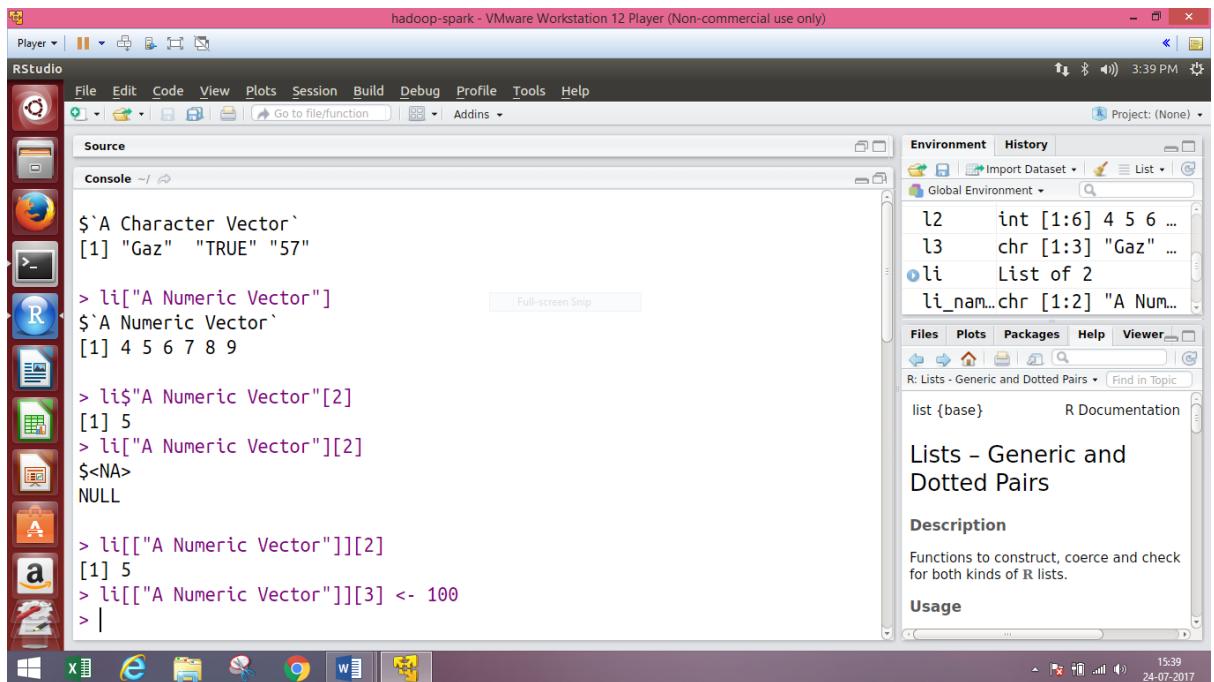
This screenshot shows the RStudio interface again. The console window contains the following R code and its execution results:

```
> li1 = list(l1,l2)  
> names(li1) = c("something","nothing")  
> li1  
$something  
    b      v      m  
TRUE FALSE TRUE  
  
$nothing  
[1] 4 5 6 7 8 9  
  
> li1[["something"]]['b']  
b  
TRUE  
>
```

Below the console, a terminal window shows the `ifconfig` command output.

The right pane of RStudio shows the help documentation for the `list` function, specifically the section on "Dotted Pairs".

Figure 71: Access list element with list and vector name



The screenshot shows an RStudio interface running on a Windows host. The console window displays the following R code and its output:

```
$`A Character Vector`
[1] "Gaz"  "TRUE" "57"

> li<-["A Numeric Vector"]
$`A Numeric Vector`
[1] 4 5 6 7 8 9

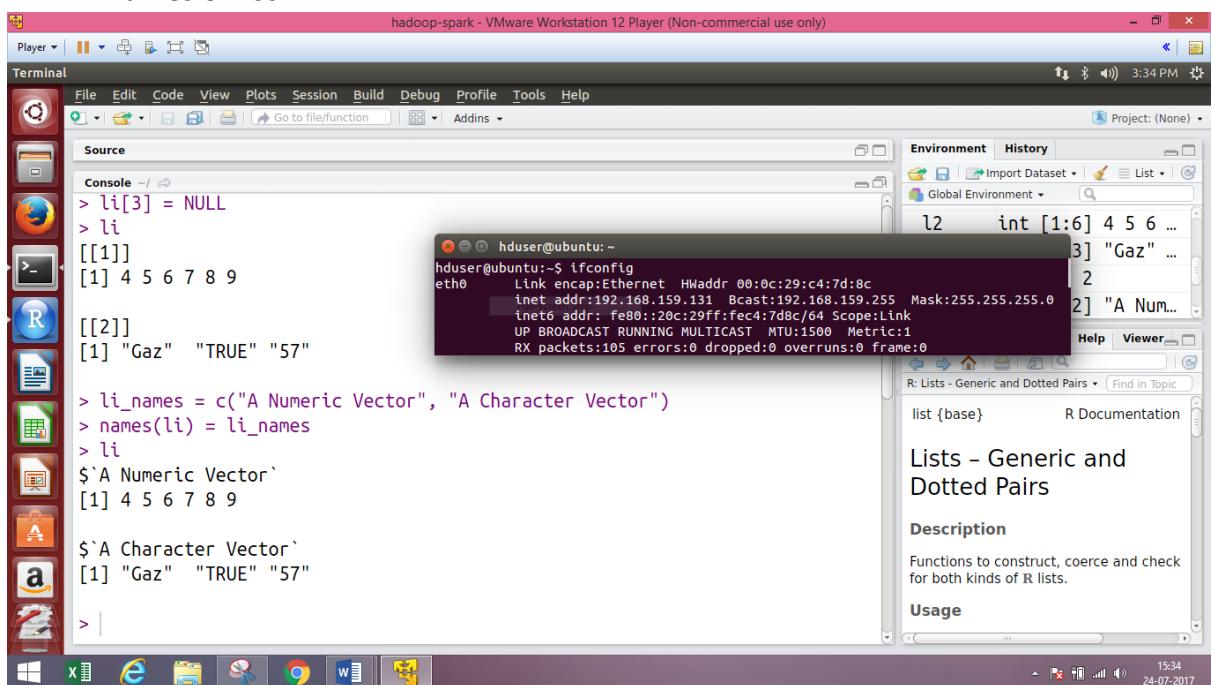
> li$"A Numeric Vector"[2]
[1] 5
> li["A Numeric Vector"][2]
$<NA>
NULL

> li[[["A Numeric Vector"]][2]
[1] 5
> li[[["A Numeric Vector"]][3] <- 100
> |
```

The right pane of RStudio shows the Global Environment, listing objects like `li` and `li_names`. The Help pane is open to the `list` documentation, specifically the section on "Lists - Generic and Dotted Pairs".

Figure 72: Access List with names

- Names of list:**



The screenshot shows an RStudio interface running on a Windows host. The console window displays the following R code and its output:

```
> li[3] = NULL
> li
[[1]]
[1] 4 5 6 7 8 9

[[2]]
[1] "Gaz"  "TRUE" "57"

> li_names = c("A Numeric Vector", "A Character Vector")
> names(li) = li_names
> li
$`A Numeric Vector`
[1] 4 5 6 7 8 9

$`A Character Vector`
[1] "Gaz"  "TRUE" "57"
> |
```

A terminal window is visible in the background, showing the output of the command `ifconfig` on an Ubuntu system, listing network interface details.

Figure 73: Naming the list

- Unlist:** See [UNLIST](#)
- Length of list:**

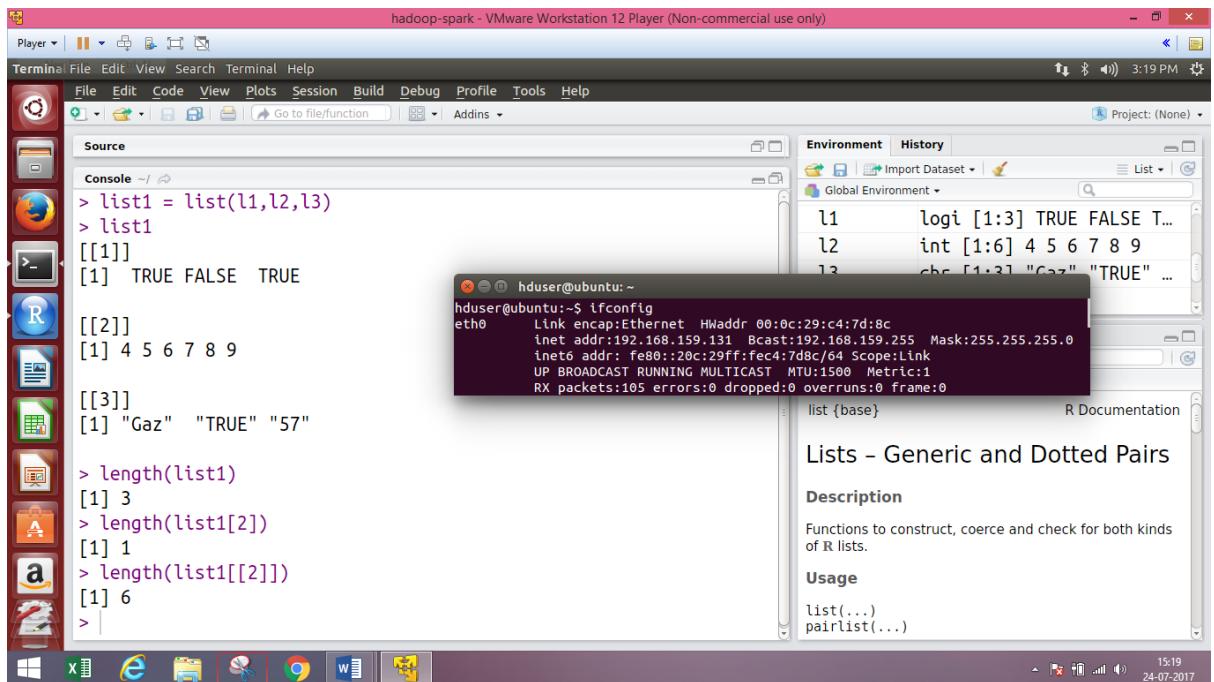


Figure 74: list length

- **List Manipulation:**

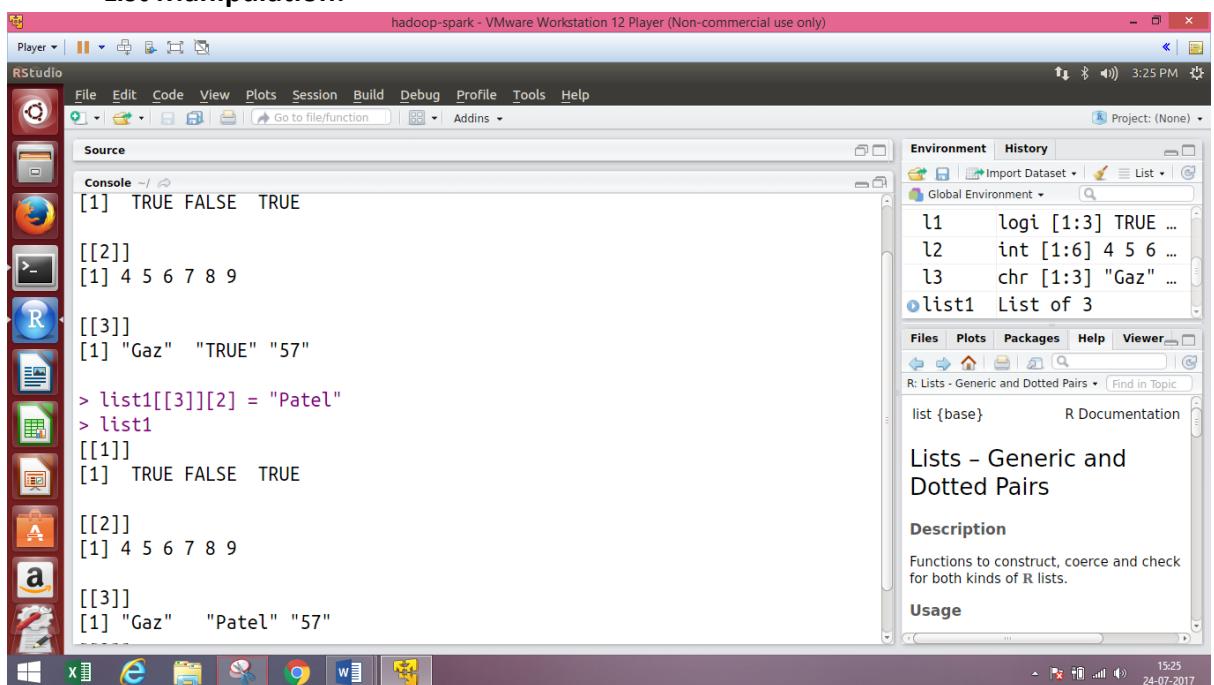


Figure 75: Modify element in List

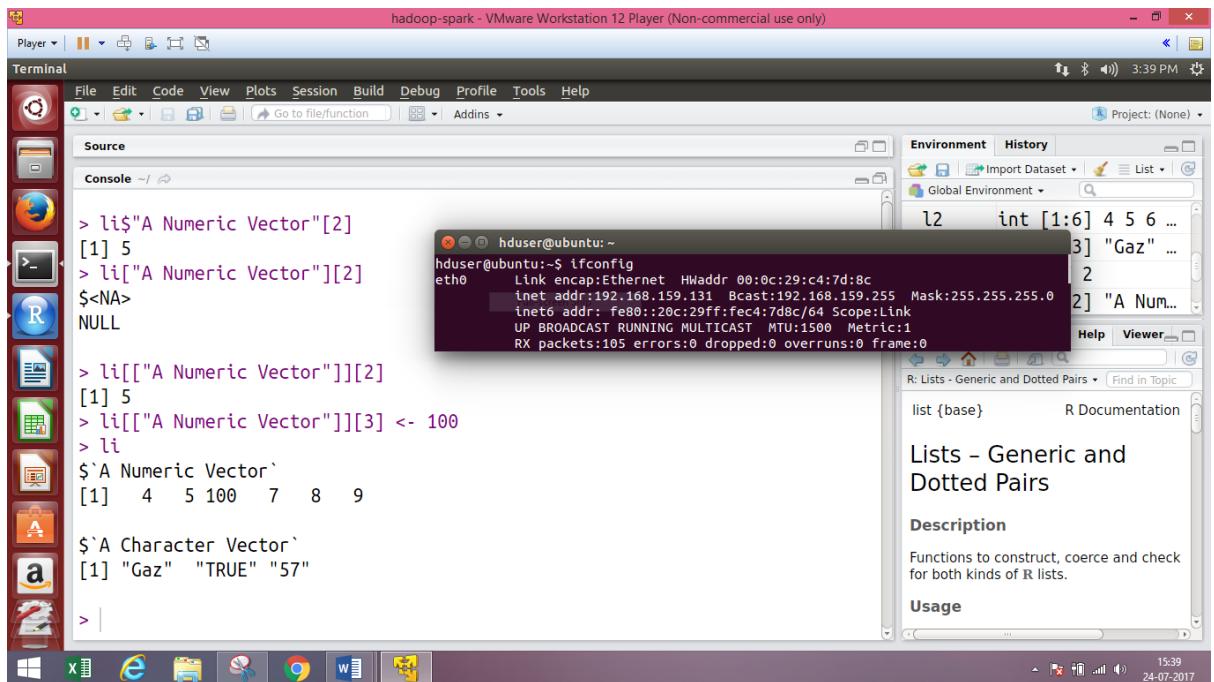


Figure 76: Modify List with Names

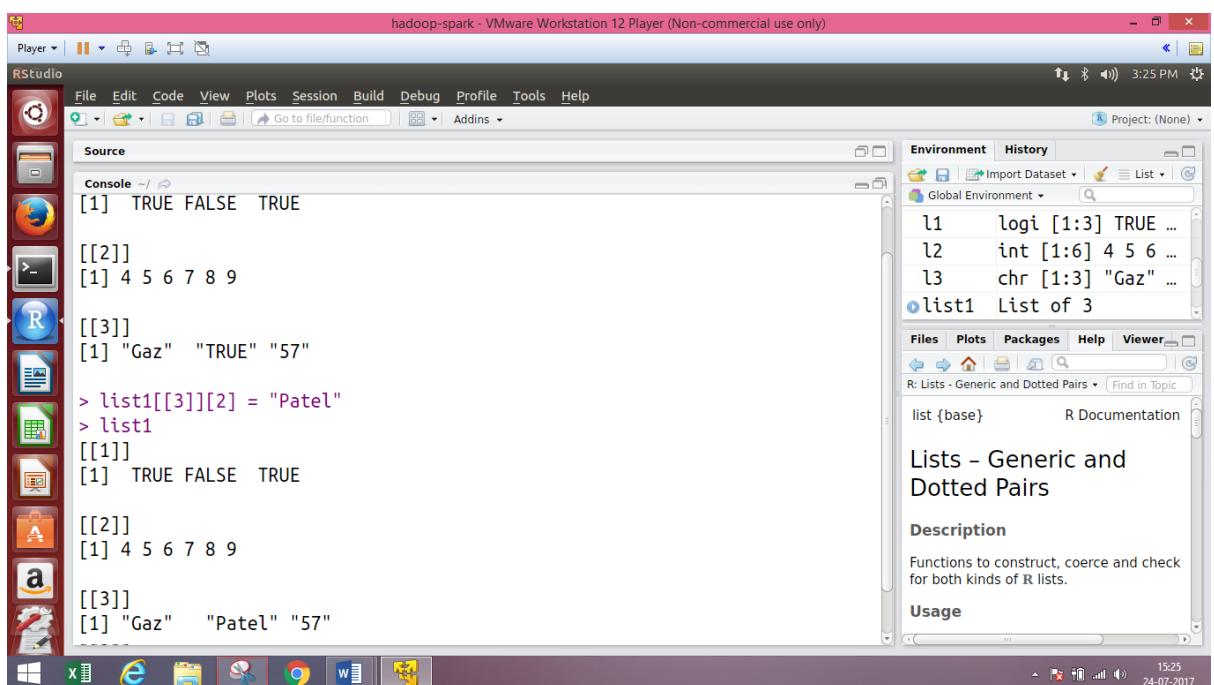


Figure 77: Add List Element

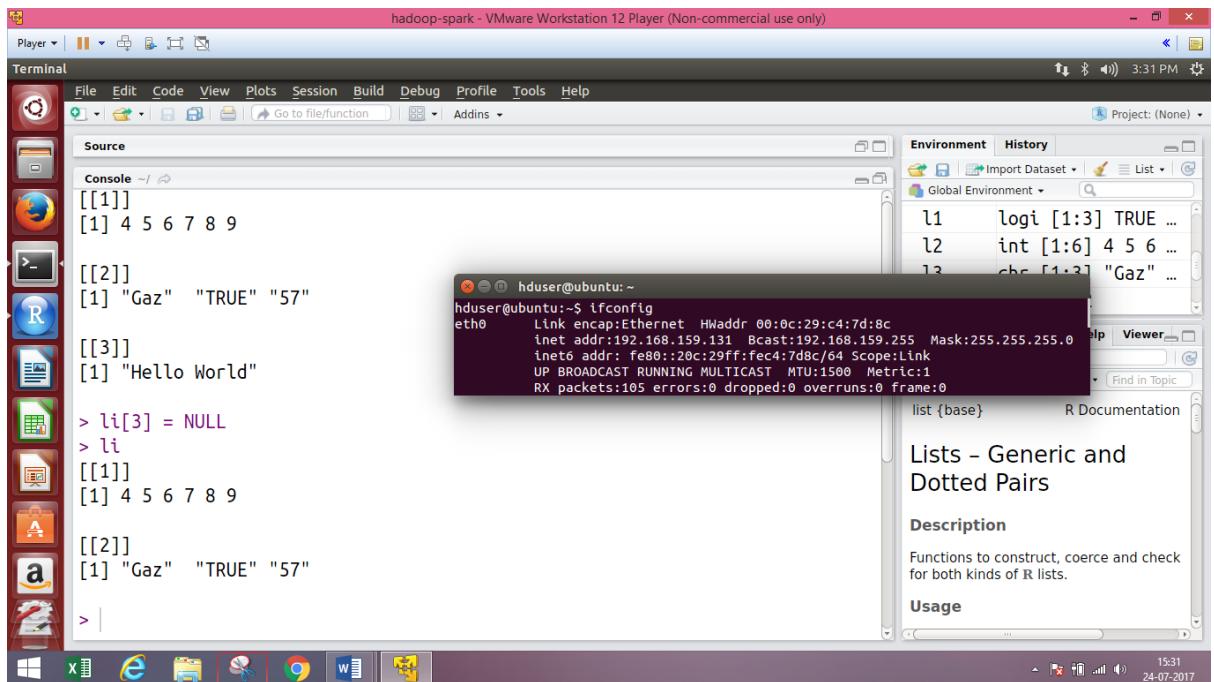


Figure 78: Delete List Element

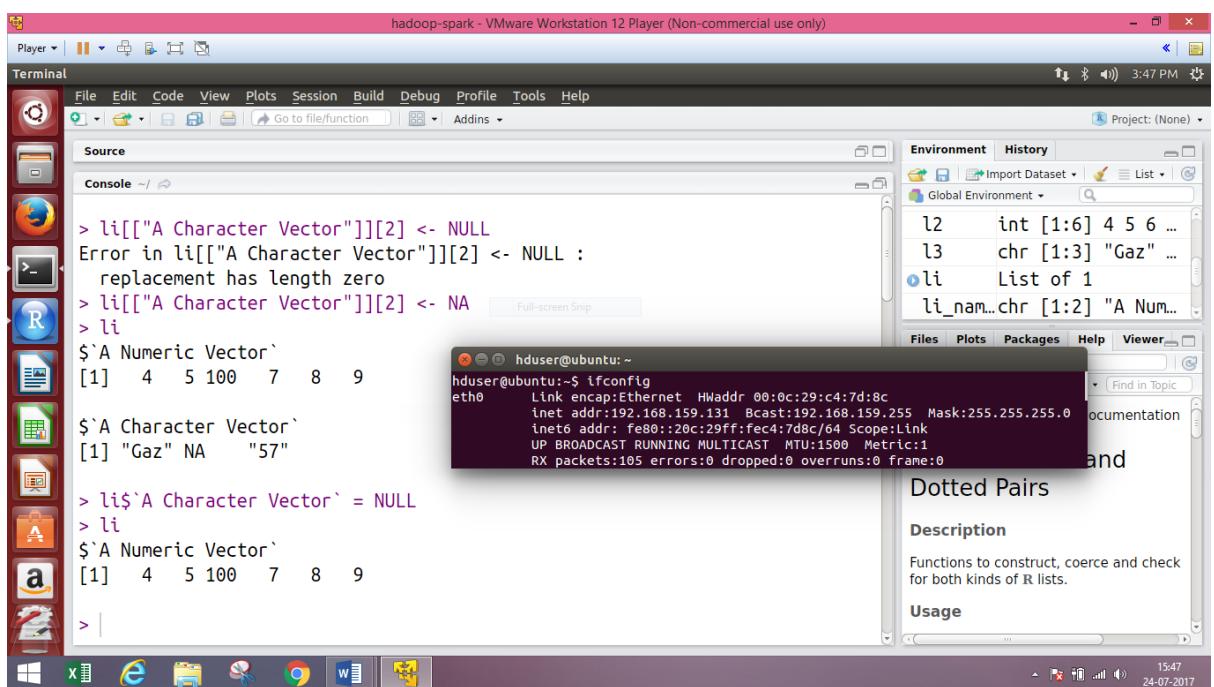


Figure 79: Delete List Element with Names

- Merge lists:

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorer, and browser. The main area has a terminal window titled 'Terminal' and a 'Console' window.

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit Code View Plots Session Build Debug Profile Tools Help
Project: (None)

Source
Console ~/
TRUE FALSE TRUE

$nothing
[1] 4 5 6 7 8 9

> li2 = c(li,li1)
> li2
$A Numeric Vector
[1] 4 5 100 7 8 9

$something
b v m
TRUE FALSE TRUE

$nothing
[1] 4 5 6 7 8 9

>

```

A tooltip for the 'c' function is open in the bottom right, showing its documentation:

Dotted Pairs
Description
Functions to construct, coerce and check for both kinds of R lists.
Usage

Figure 80: Merging Lists

10. Working with Matrix:

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorer, and browser. The main area has a terminal window titled 'Terminal' and a 'Console' window.

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit View Search Terminal Help
Project: (None)

df* c* var* Untitled1*
Source on Save Run Source
1 rnames = c("r1","r2")
2 cnames = c("c1","c2")
3 mymat = matrix(c(5:8),nrow = 2,byrow = T,dimnames = list(rnames,cnames))
4 print(mymat)
5 print(mymat[,1])
6 print(mymat[,2])
7 print(mymat[,2]%%2==0)
8
6:17 (Top Level) ~

> rnames = c("r1","r2")
> cnames = c("c1","c2")
> mymat = matrix(c(5:8),nrow = 2,byrow = T,dimnames = list(rnames,cnames))
> print(mymat)
  c1 c2
r1  5  6
r2  7  8
> print(mymat[1,2])
[1] 6

```

A tooltip for the 'matrix' function is open in the bottom right, showing its documentation:

Description
Performs **set union**, intersection, (asymmetric!) difference, equality and membership on two vectors.
Usage

Figure 81: Create Matrix

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorer, and browser. The main area has tabs for 'Untitled1*', 'df.x', 'c.x', and 'var.x'. The 'Console' tab is active, displaying R code and its output. The code creates two character vectors 'rnames' and 'cnames', a matrix 'mymat' with dimensions 5x8, and prints them. A terminal window is open in the background, showing the output of the 'ifconfig' command on an Ubuntu system.

```

> rnames = c("r1", "r2")
> cnames = c("c1", "c2")
> mymat = matrix(c(5:8), nrow = 2, byrow = T, dimnames = list(rnames, cnames))
> print(mymat)
   c1 c2
r1  5  6
r2  7  8
> print(mymat[1,2])
[1] 6
> print(mymat[,2])
r1 r2
 6  8
> print(mymat[,2]%%2==0)
r1 r2
TRUE TRUE
>

```

Figure 82: Access Matrix

11. Working with Randoms: See [runif\(\)](#) and [seed\(\)](#) functions.

12. Programs:

- Write a program in R where you would initialize a vector with integers ranging from 1 to 10, take a sample from that vector and check if the number is greater than 7 or not.

The screenshot shows the RStudio interface running on a Windows host. The left sidebar contains icons for various applications like R, file explorer, and browser. The main area has tabs for 'Untitled1*', 'df.x', 'c.x', and 'var.x'. The 'Console' tab is active, displaying R code and its output. The code initializes a vector 'v1' with values 1 to 10, samples one element 's1' from it, checks if it's greater than 7, and prints the result. A terminal window is open in the background, showing the output of the 'ifconfig' command on an Ubuntu system.

```

1 v1 = 1:10
2 s1 = sample(v1)
3 l1 = any(s1>7)
4 cat("If the number is greater than 7 or not: ",l1)
5

> v1 = 1:10
> s1 = sample(v1)
> l1 = any(s1>7)
> cat("If the number is greater than 7 or not: ",l1)
If the number is greater than 7 or not:  TRUE
>

```

Figure 83: Program 1

- Write a program to initiate two objects and swap their contents

Basic Programming Assignment-2

Gazal Patel - DSFT1701173

The screenshot shows the RStudio interface running on a Windows host. On the left is a vertical toolbar with icons for various applications. The main window has tabs for 'Terminal' and 'Console'. In the 'Terminal' tab, the code is displayed:

```
B1 = 5
B2 = 7
cat("Number1: ",B1," Number2: ",B2)
B2 = B1+B2
B1 = B2-B1
B2 = B2-B1
print("\n--After Swapping--\n")
cat("Number1: ",B1," Number2: ",B2)
```

Below the code, a terminal window shows the output of the 'ifconfig' command:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131  Bcast:192.168.159.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4c:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
```

The 'Console' tab shows the results of the R code execution:

```
B1 = 5
B2 = 7
cat("Number1: ",B1," Number2: ",B2)
Number1: 5 Number2: 7
> B2 = B1+B2
> B1 = B2-B1
> B2 = B2-B1
> print("\n--After Swapping--\n")
[1] "\n--After Swapping--\n"
> cat("Number1: ",B1," Number2: ",B2)
Number1: 7 Number2: 5
>
```

The 'Environment' pane on the right shows the values of B1 and B2.

Figure 84: Program 2 - Method1- (1/2)

This screenshot continues from Figure 84. The 'Console' tab now shows the results of the R code execution after the swap:

```
B1 = 5
B2 = 7
cat("Number1: ",B1," Number2: ",B2)
Number1: 5 Number2: 7
> B2 = B1+B2
> B1 = B2-B1
> B2 = B2-B1
> print("\n--After Swapping--\n")
[1] "\n--After Swapping--\n"
> cat("Number1: ",B1," Number2: ",B2)
Number1: 7 Number2: 5
>
```

The terminal window still displays the 'ifconfig' output. The 'Environment' pane on the right shows the values of B1 and B2.

Figure 85: Program 2 - Method 1 - (2/2)

Basic Programming Assignment-2

Gazal Patel - DSFT1701173

```

n1 = "Gazal"
n2 = "Patel"
cat("object 1: ",n1," object 2: ",n2)
temp_var = n1
n1 = n2
n2 = temp_var
cat("object 1: ",n1," object 2: ",n2)

```

```

> n1 = "Gazal"
> n2 = "Patel"
> cat("object 1: ",n1," object 2: ",n2)
object 1: Gazal object 2: Patel
> temp_var = n1
> n1 = n2
> n2 = temp_var
> cat("object 1: ",n1," object 2: ",n2)
object 1: Patel object 2: Gazal
>

```

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13200 (12.9 KiB) TX bytes:14400 (14.0 KiB)


```

Figure 86: Program 2 - Method 2

- Initialize 2 vectors ‘a’ and ‘b’ with 5 numbers each, divide each elements of vector ‘a’ with that of vector ‘b’ and store the result into a new vector.

```

a = c(10,20,30,40,5,10,15,20)
b = c(1,2,3,4)
ans = a/b
cat("Answer of a/b is: ",ans)

```

```

> a = c(10,20,30,40,5,10,15,20)
> b = c(1,2,3,4)
> ans = a/b
> cat("Answer of a/b is: ",ans)
Answer of a/b is: 10 10 10 10 5 5 5 5
>

```

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.131 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13200 (12.9 KiB) TX bytes:14400 (14.0 KiB)


```

Figure 87: Program 3