

**Foundations of R Software**  
**Prof. Shalabh**  
**Department of Mathematics and Statistics**  
**Indian Institute of Technology, Kanpur**

**Lecture - 32**  
**Vector Indexing**

Hello friend, welcome to the course Foundations of R Software. You can recall that in the last couple of lectures we are talking about the tools for the data management and we have learnt a couple of commands for the data manipulations. So, in this lecture also we are going to continue on the same broader topic and we are going to talk about the indexing of the vector that is Vector Indexing.

So, the first question comes here; what is a vector index? Can you recall that whenever you try to look into your books, after the cover page the first thing is index. What does this index give you? First of all on the left hand side usually there are the topics which are the contents of the book and on the, right hand side, there is a page number corresponding to every topic.

So, similar is the operation in the data vectors also, that when we have a data vector we have the values of the data vector they are located on the first, second, third, fourth positions. So, now, we need to understand how we can correspond between the values of the variable and the location of those values and beside those things we will try to understand some other very small topics which are very important for the management of data tools. So, let us begin our lecture and try to understand.

**Vector Indexing**

**Vector and its values**

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

*Handwritten: 1, 2, ..., 10*

```
> x[x > 5]
[1] 6 7 8 9 10
```

*Handwritten: TRUE*

```
> x[(x%%2==0)]
[1] 2 4 6 8 10
```

*Handwritten: TRUE*

```
> x[(x%%2==1)]
[1] 1 3 5 7 9
```

*Handwritten: TRUE*

```
> x = 1:10
> x[(x > 5)]
[1] 6 7 8 9 10
```

```
> x[(x%%2==0)]
[1] 2 4 6 8 10
```

```
> x[(x%%2==1)]
[1] 1 3 5 7 9
```

*Handwritten: which are those values for which x > 5 says TRUE*

*Handwritten: #%% indicates x mod y #values for which x mod 2 is 0*

*Handwritten: #values for which x mod 2 is 1*

*Handwritten: 1 2 3 4 5 ... F T F T ...*

So, first let me try to give you a very simple operation related to the vector. So, suppose I try to consider here a data vector which has got the values 1 to 10 like this,, right, this is here  $x$ . Now, I just want to show you a couple of operations which are related to this vector. So, now, suppose if you want to know that how many values in this data vector are more than 5 and what are those values,, right.

So, in case if you want to know that which values are more than 5 so, for that you can use here a command like here  $x$  greater than 5. So, this is going to give you an outcome like here for the values 1 2 5 it will say here false and for 6 to 10 it will say here true. now, your objective is not really to find out only the true and false, but you want to know that which are those values which are greater than 5 or in simple words which are those values for which  $x$  greater than 5, says true, right.

So, now if you want to do it the rule is very simple just try to write down the name of the data vector and write down the square brackets and enclose the condition inside the square. So, if you try to see here when I am trying to write down here  $x$  greater than 5 that is going to give the outcome as TRUE or FALSE.

But now I am trying to do here is the following, that what are the values in this data vector  $x$  which are greater than 5 or which are the values of this data vector consisting of TRUE and FALSE in this  $x$  greater than 5, what are the values corresponding to which we have TRUE. and it will give you the answer here 6 7 8 9 10.

So, you can see here that these are the values here 6 7 8 9 10 which are more than 5. Similarly, if you want to know that which of the values in this data vector  $x$  are even or say odd, in that case what we can do for each of the values in this data vector  $x$  we can have the modulo operation modulo division by 2 and in case if the remainder comes out to be here 0 then we have to find which are those values corresponding to which we have the remainder 0.

So, when you try to write down here this  $x$  modulo division 2 is exactly equal to 0 once again this is going to give you the answer in terms of TRUE and FALSE. And you need to find out what are the values corresponding to which we have the TRUE outcome or the outcome as TRUE T R U E logical TRUE and then when you try to write down here  $x$  and try to write down this logical condition inside the square bracket this will give you

what are the values in this data vector x corresponding to which the modulo division with 2 gives the remainder 0.

And this comes out to be here like this 2 4 6 8 10. So, what will happen? It will go for 1 2 3 4 5 and so on when it goes to 1 answer comes out to be here FALSE, then it goes to 2 answer comes out to be here TRUE, then it goes to 3 the answer comes out to be here FALSE, for TRUE the answer comes out to be here TRUE and so on and then FALSE values will not come here only the TRUE values will be reported here or the TRUE is coming corresponding to a values 2 4 6 8 and 10.

And similarly, if you want to know about the out so, you simply try to do the modulo division with respect to 2 and you want to find out those values corresponding to which we have the outcome as 1 that is the remainder is 1. So, you try to write down here x and try to see that what are the values in this x for which the modulo division with 2 is giving you an answer as FALSE F A L S E logical FALSE and you try to write down this condition inside the square bracket and you get here like this here 1 3 5 7 9.

So, all those values whose remainder is coming out to be 1 they will be reported here out of the vector, ok.

(Refer Slide Time: 05:49)

```
Vector Indexing  
□ Vector with missing observation(s)  
> x[5] = NA  
> x  
[1] 1 2 3 4 NA 6 7 8 9 10  
> y = x[!is.na(x)] #! Means negation  
> y  
[1] 1 2 3 4 6 7 8 9 10 # 5 is missing  
> mean(x)  
[1] NA  
> mean(y)  
[1] 5.555556
```

Similarly, I try to show you here one more operation and then after that I will try to show you these things on the R software. Suppose, I try to replace the 5th value in the data vector by NA so, here I am trying to inform you here one more operation that if you have

a data vector x and if you want to access any particular value then you have to write the square bracket and inside the square bracket you have to write the location index.

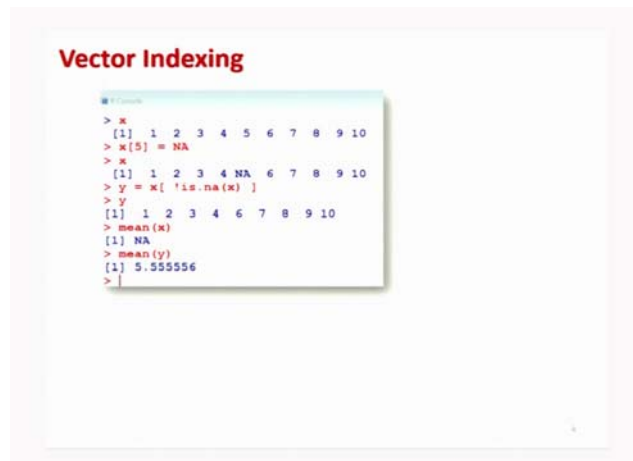
For example, if I want to access the 5th value in the data vector x I will write down here x and inside the square bracket as 5. So, now, you can see here you are trying to replace the 5th value in the data vector x by NA. So, now, you can see here this data vector is 1 2 3 4 and then 5th value 5 is replaced by NA and then we have a 6 7 8 9 10. Now, you try to do logical operation here and you want that the missing values are removed from this data vector and all the available values are stored in a data vector y.

So, what I try to do here that in order to know that which are the values, you can see here I am writing here this symbol a symmetry sign that was the negation logical dot is dot na, na inside the parenthesis x, right and you know that is dot na is a command to find out is there any missing values NA in the data vector x and you are trying to say that negation; no, you do not want.

So, whatever are the values which are available which are not missing in the data vector x they will be reported here, you can see here this outcome will come out to be here 1 2 3 4 and then 6 7 8 9 10 and 5 here is missing.

So, now the advantage is that if you try to find out the mean of x which has got here NA. So, you will not get here any value you will get only here NA, but in case if you try to find out the mean of the values in the data vector y this will come out to be here like this 5.55, right.

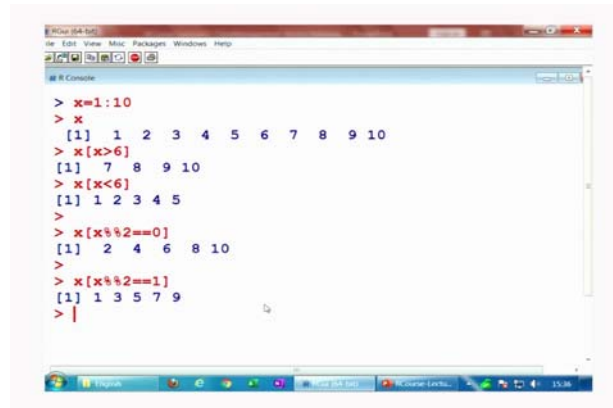
(Refer Slide Time: 07:50)



```
Vector Indexing
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[5] = NA
> x
[1] 1 2 3 4 NA 6 7 8 9 10
> y = x[ !is.na(x) ]
> y
[1] 1 2 3 4 6 7 8 9 10
> mean(x)
[1] NA
> mean(y)
[1] 5.555556
> |
```

So, and you can see here this is the operation on the R console. So, let us try to first see these operations on the R console and then we will try to see here, right.

(Refer Slide Time: 08:03)

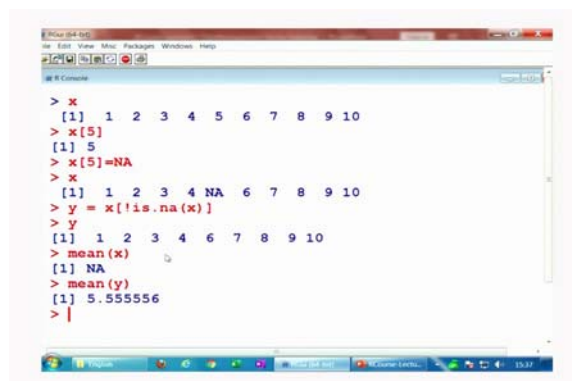


```
> x=1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[x>6]
[1] 7 8 9 10
> x[x<6]
[1] 1 2 3 4 5
>
> x[x%%2==0]
[1] 2 4 6 8 10
>
> x[x%%2==1]
[1] 1 3 5 7 9
> |
```

So, now I try to create here a data vector here x, x here is like this and suppose I want to know here that what are the values in x suppose which are greater than 6 this will give you here 7 8 9 10. And similarly, if I want to know here what are the values which are smaller than 6 you can see here 1 2 3 4 5, right. And similarly, if you want to know about this even odd then what I can do here, I would like to find out those values of here x for which the modulo division with 2 is giving me an answer which is logically equal to 0.

So, all those values when divided by 2 if they are giving the remainder 0; that means, that the number is even and similarly if you want to know what are the odd values. So, you can have the modulo division with 2 and try to see where the remainder is 1 and these are this value 1 3 5 7 9, right.

(Refer Slide Time: 08:59)



```
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[5]
[1] 5
> x[5]=NA
> x
[1] 1 2 3 4 NA 6 7 8 9 10
> y = x[!is.na(x)]
> y
[1] 1 2 3 4 6 7 8 9 10
> mean(x)
[1] NA
> mean(y)
[1] 5.555556
> |
```

And now this is your here x and if you want to x is any particular value say I want to access here 5th value. So, this is here 5 and in case if you want to replace this 5th value by here NA. So, you can see here because here is like this and now x become here like this.

So, now I want to store here all the values in x which are dot equal to NA. So, I try to write down here negation sign is dot na and then here x like this. So, you can see here y contains here all the values except the 5th value which is here NA and now if you try to find out here mean of here x it will give you here NA, but if you try to find out the mean of y this will give you here this value.

So, you can see here this is very useful operation and it will try to help you in searching a particular type of value and the condition for searching can be given in the square brackets and just outside the square brackets try to see the vector in which you want to do the search operation.

(Refer Slide Time: 09:57)

**Vector Indexing**

□ Vector of negative integers

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[-(1:5)]
[1] 6 7 8 9 10
has the same outcome as
> x[(6:10)]
[1] 6 7 8 9 10
```

Inset window output:

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[-(1:5)]
[1] 6 7 8 9 10
> x[(6:10)]
[1] 6 7 8 9 10
> |
```

And besides those things you can do some other type of operation which I will try to show you with this operation.

Now, just about the indexing I want to show you here that, how do you index? So, if you try to take here the same data vector is from 1 to 10 then what is this indicating x inside the square bracket minus 1 colon 5?

So, it is something like here x see here -1, -2, -3, -4 and here -5, right. So, what is happening here? So, it is trying to read from the end and it is trying to go in the negative direction because it is here minus. And then it is trying to count here the last 5 values which are here 6 7 8 9 10 and it is giving you here.

So, this is the same outcome that if you try by writing x inside the square bracket from 6 to 10 that is 6 colon 10. So, this will give you here the outcome 6 7 8 9 10, right. So, you can see here that the same outcome can be obtained by the two different options, but surely you have to keep in mind the mathematics behind it that how it is working, right.

(Refer Slide Time: 11:12)

**String vector**

The elements of a vector can be named.

Using these names, we can access the vector elements.

`names` is used for functions to get or set the names of an object

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
$a1
[1] 1
$a2
[1] "c"
$a3
[1] 1 2 3

> names(z)
[1] "a1" "a2" "a3"
```

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
$a1
[1] 1
$a2
[1] "c"
$a3
[1] 1 2 3
> names(z)
[1] "a1" "a2" "a3"
```

Now, let me try to give you here one more example and this operation is about changing the names in the list, right. So, names is a function here for example, n a m e s this is used for functions to get the or to get the names of an object. What really happened that when you are trying to give some values inside a data vector or a list then R automatically give it's a name, right and you want to change the name. Then how to get it done? Once you change the name then you can access those values by the name instead of the index.

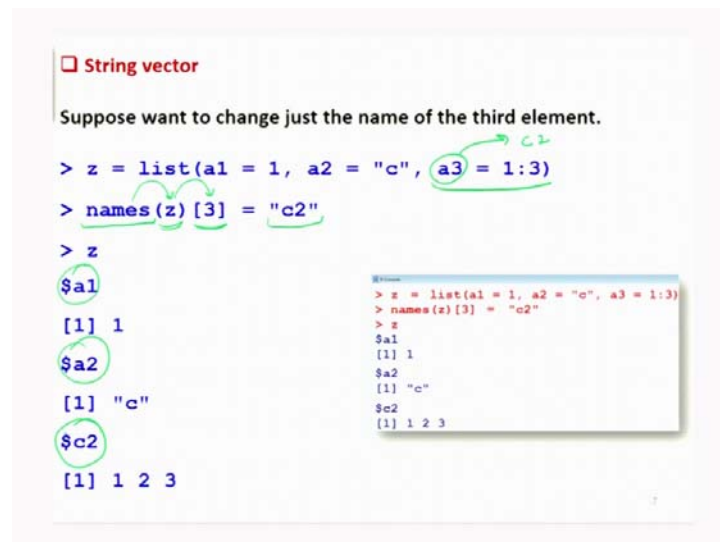
So, for example, I try to consider here a list where I am trying to create here a list with the elements which is number a 1 equal to 1, the second value a 2 is equal to "c" which is a string character and third value here a 3 as the numbers 1 2 3, right. So, now, if you try to see here you are trying to give it here a name say a 1, a 2, a 3, right and earlier if you

try to see if you do not give any name possibly it was using like this double square bracket.

So, now if you try to see the construct of the name this is here like this z this is 1 “c” and 1 2 3 and the first value has got the name a 1, the second value has got the name a 2 and third value has got the name a 3 as you have given in the list.

Now, in case if you want to know that what are the names in the list, somebody has given you the list and you want to know the name, name of all the objects in the list. So, you simply have to write down here names n a m e s and then parenthesis you try to write down the list. So, this will give you here like this “a 1”, “a 2” and “a 3” which are your strings and this is the same operation that you can do on the R console also and this is the screenshot.

(Refer Slide Time: 13:09)



```
String vector
Suppose want to change just the name of the third element.
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> names(z)[3] = "c2"
> z
$a1
[1] 1
$a2
[1] "c"
$c2
[1] 1 2 3
```

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> names(z)[3] = "c2"
> z
$a1
[1] 1
$a2
[1] "c"
$c2
[1] 1 2 3
```

Now, I want to change the name, suppose I want to change the name of the third object. So, third object here is like this here and whose name is a 3 and suppose I want to change this name as a “c 2”. So, how to get it done? Very simple, try to understand how I am doing it, just trying to write down the about names and n a m e s and then in the parenthesis try to write down the name of the object which is here z.

Now, you want to access the third element. So, you try to write down here is square brackets and then inside the square bracket you write 3. So, this is now indicating that



you are trying to find out the names of the object z and then you are trying to find out the name of the third value in the z.

So; obviously, this will be here something, but you are trying to assign it a new name say “c 2”. So, you try to write down this name within the double quote because this is a character and now you see what happens. This in the z first and second names remain the same, but the third name is change here as “c 2”, right.

(Refer Slide Time: 14:22)

```
String vector
Example
names is used for functions to get or set the names of an object
> x = c(water=1, juice=2, lemonade=3 )
> names(x)
[1] "water" "juice" "lemonade"
> x["juice"]
juice
2
```

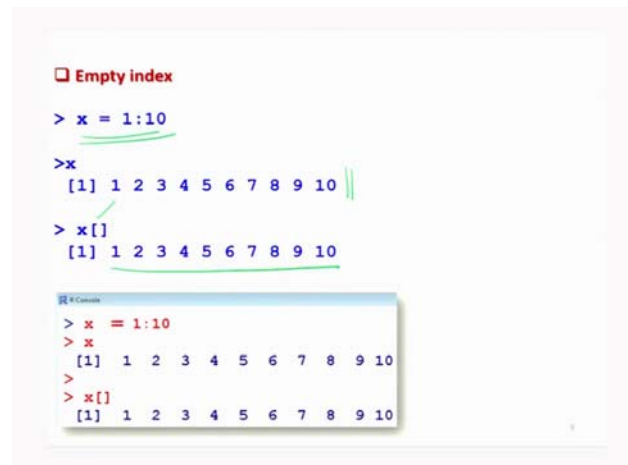
The screenshot shows an R console session. The first line is a title "String vector" with a red square icon. Below it is "Example" in red. The text "names is used for functions to get or set the names of an object" is in blue. The code is as follows: `> x = c(water=1, juice=2, lemonade=3 )`, `> names(x)` returns `[1] "water" "juice" "lemonade"`, and `> x["juice"]` returns `juice` and `2`. Handwritten green circles and arrows highlight the "juice" name in the first command, the "juice" element in the second command's output, and the value "2" in the third command's output. A separate terminal window on the right shows the same sequence of commands and outputs.

And similarly, in case if you want to do that you are given the names and corresponding to the names you want to know the value then how to get it done. So, let me try to take here one more example in which I am trying to take here 3 values say water equal to 1, juice is equal to 2 and lemonade equal to 3, right.

So, now you can see here intentionally I am not taking here list, but I am simply trying to take here data vector here c. So, if you try to see here the names in this vector x are like as “water” “juice” and “lemonade” and now you want to know in this data vector x what is the value of the juice.

So, you try to write down here the name “juice” inside the square bracket and then try to write what is the value inside this data vector x corresponding to which the name is juice and if you try to enter here this will give you here value 2. This value 2 is this value here and this is here the screenshot of the same operation, right.

(Refer Slide Time: 15:26)



```
Empty index

> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[]
[1] 1 2 3 4 5 6 7 8 9 10
```

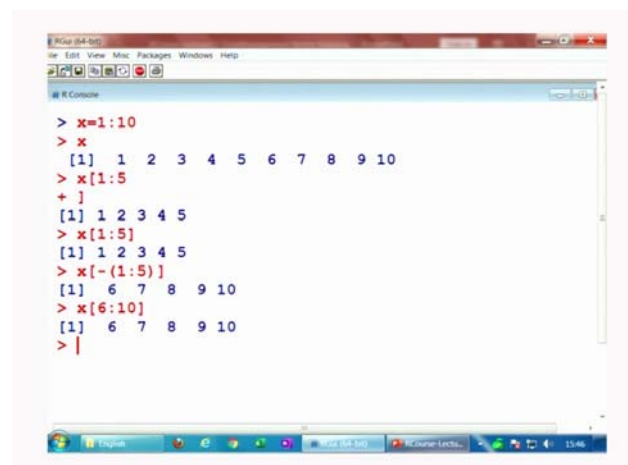
The screenshot shows an R console window with a title bar 'RGui (64-bit)'. The console output is as follows:

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[]
[1] 1 2 3 4 5 6 7 8 9 10
```

So, and then you have one more very simple operation that when you are trying to write down here x equal to 1 colon 10 it will give you this output, but if you simply write down here x and then here after this if you simply write down here x and then this square bracket here this will give you the same outcome. So, depending on your need means you can use this arguments and your data manipulation will become simple.

So, now, let me try to show you these commands on the R console here, so that you get here more confident.

(Refer Slide Time: 15:58)



```
RGui (64-bit)
File Edit View Misc Packages Windows Help

R Console

> x=1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[1:5]
+ ]
[1] 1 2 3 4 5
> x[1:5]
[1] 1 2 3 4 5
> x[-(1:5)]
[1] 6 7 8 9 10
> x[6:10]
[1] 6 7 8 9 10
> |
```

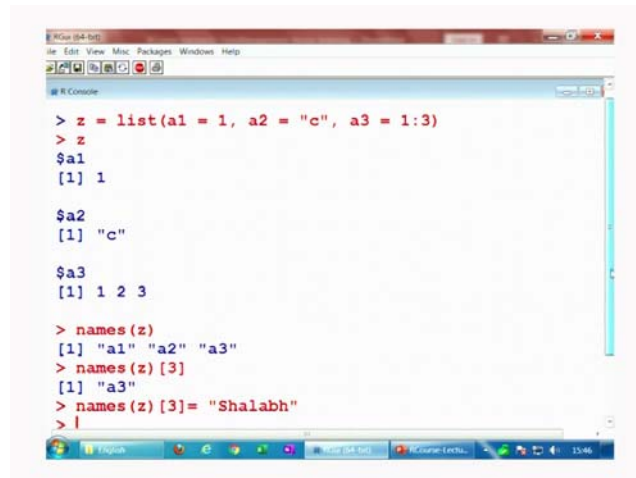
The screenshot shows an R console window with a title bar 'RGui (64-bit)'. The console output is as follows:

```
> x=1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[1:5]
+ ]
[1] 1 2 3 4 5
> x[1:5]
[1] 1 2 3 4 5
> x[-(1:5)]
[1] 6 7 8 9 10
> x[6:10]
[1] 6 7 8 9 10
> |
```

So, I try to create here one data vector here say here 1 to 10 which is here like this. Now, if you want to see here what will the value from here 1 to 5 you can see here this is going to be a 1 2 3 4 5 or yeah you can write it directly here like this, but now if I try to write

down here say minus times this 1 to 5 you see what happens like this, right. So, and this is same as if you try to write down here 6 to 10, right, there is no problem in this. So, now, I try to create list.

(Refer Slide Time: 16:31)



```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
$a1
[1] 1

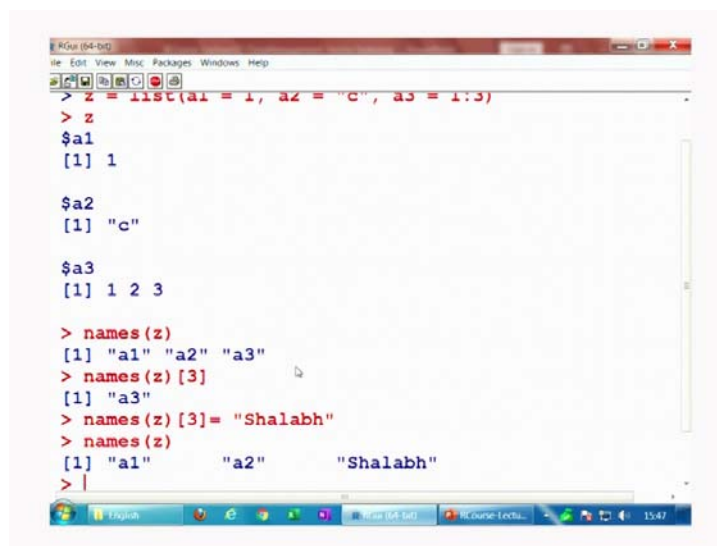
$a2
[1] "c"

$a3
[1] 1 2 3

> names(z)
[1] "a1" "a2" "a3"
> names(z)[3]
[1] "a3"
> names(z)[3] = "Shalabh"
> |
```

So, this is your here list and if you want to know the what are the names in this z you can see here like this, right. And in case if you want to change the names at the third place you can see here names z and if you try to see here 3 what is this coming out to be here, "a 3" and now you try to write down here that I want to give it here a name say here "Shalabh" and enter here, right.

(Refer Slide Time: 16:56)



```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
$a1
[1] 1

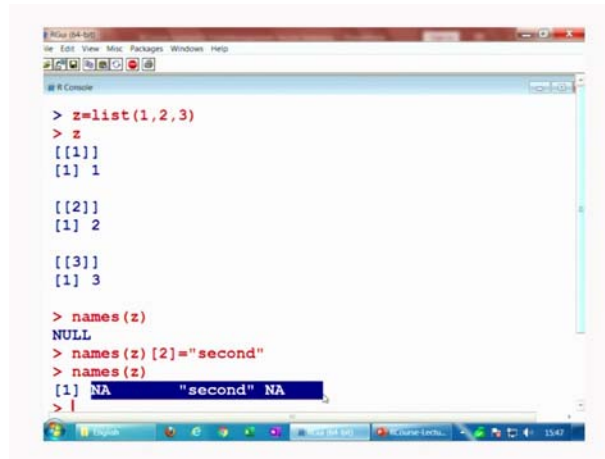
$a2
[1] "c"

$a3
[1] 1 2 3

> names(z)
[1] "a1" "a2" "a3"
> names(z)[3]
[1] "a3"
> names(z)[3] = "Shalabh"
> names(z)
[1] "a1"      "a2"      "Shalabh"
> |
```

And then you try to find out here the names of here z you can see here what happens here “a 1” “a 2” and “Shalabh”. So, this is how you can operate with the names, right.

(Refer Slide Time: 17:07)



```
> z=list(1,2,3)
> z
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> names(z)
NULL
> names(z)[2]="second"
> names(z)
[1] NA "second" NA
> |
```

And in case if you try to take here a simple here list like as here 1, 2 and 3. So, you can see here this is your here like this and if you try to see here names of here z they are simply here null because you have not given it any name these are the default indexes which are taken here, right, but in this case if you try to see here you want to give the second value a name say here say “second”. So, you can see here now the names will be here like as NA “second” NA, right.

(Refer Slide Time: 17:43)

**Mixed mode**  
List can be heterogeneous (mixed modes).

We can start with a heterogeneous list, give it dimensions, and thus create a heterogeneous list that is a mixture of numeric and character data:

**Example**

```
> ab = list(1, 2, 3, "X", "Y", "Z")
> dim(ab) = c(2,3)
> print(ab)
      [,1] [,2] [,3]
[1,] 1    3    "Y"
[2,] 2    "X" "Z"
```

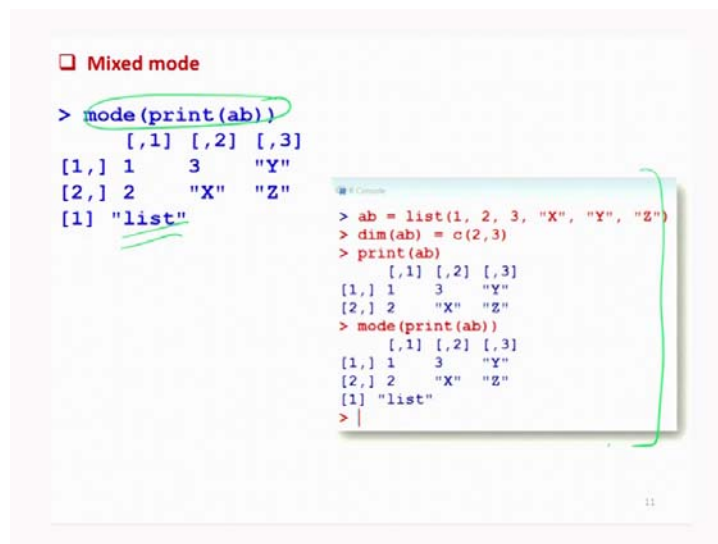
10

So, this type of means practice you can do yourself and I will try to show you here something more. Well, so, if you have supposed got a list which has got the mixed mode mixed mode means you have you do not have all the data values of the same mode like as all say numeric or all the characters.

So, in this case if I you have if you want to arrange them in the form of a matrix, it is not actually matrix because matrix is something like means a mathematical operator, but what I am trying to say here that you want to arrange the values in some rows and columns then how can you do it, right. So, let me try to show you this with an example suppose I want to create here suppose I create here a list of the 6 values 1, 2, 3 and say "X", "Y", "Z" and suppose I try to give it a name here ab and then I use here dimension of ab are equal to c 2, 3.

So, you are trying to inform ab that this values that the values which are here in this ab they have to be arranged in two rows and so you can see here just like the operation that happens in the matrix similar type of operation will happen here that 1 to 2 it will come then goes to 3, then comes to here 4, then comes to here 5th value and then to 6th value. So, it will be like here 1, 2, 3, "X" "Y" "Z", right. So, this is but remember one thing this is not a matrix.

(Refer Slide Time: 19:12)



```
❑ Mixed mode

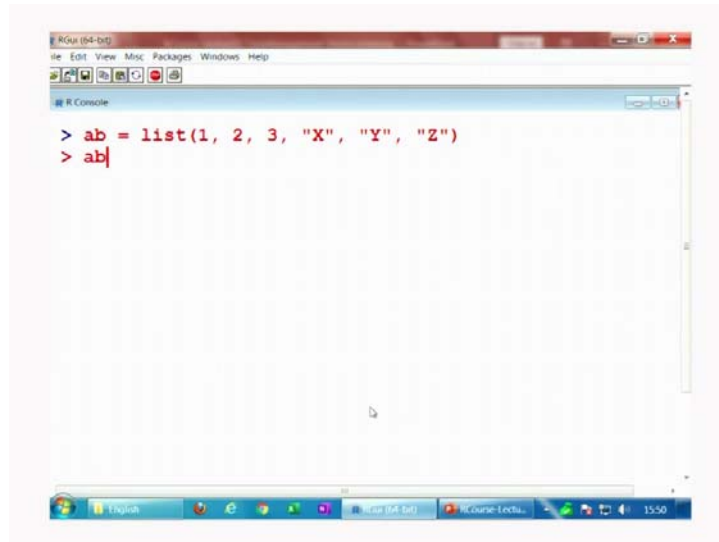
> mode(print(ab))
      [,1] [,2] [,3]
[1,] 1    3   "Y"
[2,] 2   "X"  "Z"
[1] "list"

> ab = list(1, 2, 3, "X", "Y", "Z")
> dim(ab) = c(2,3)
> print(ab)
      [,1] [,2] [,3]
[1,] 1    3   "Y"
[2,] 2   "X"  "Z"
> mode(print(ab))
      [,1] [,2] [,3]
[1,] 1    3   "Y"
[2,] 2   "X"  "Z"
[1] "list"
> |
```

What is the mode of this print ab this is here "list" and these are the operation which you will do on the here this one, right. So, now, I try to show you this operation also on the R

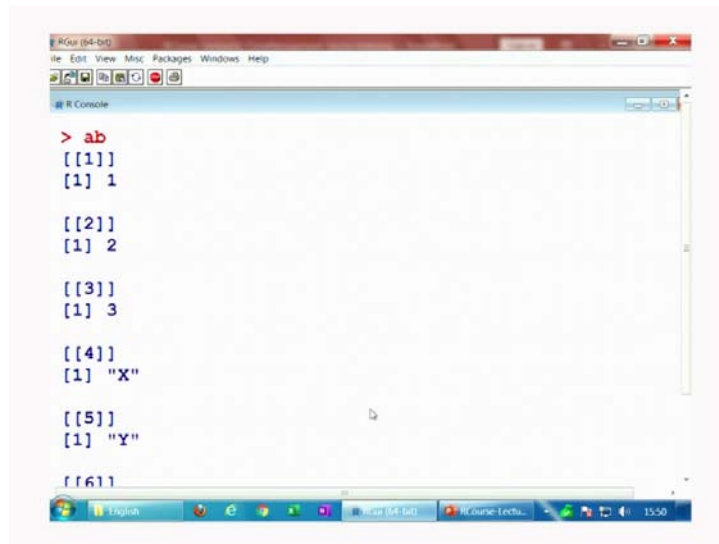
console. So, that you can be confident that these things are working this is your here ab you can see here like this, right.

(Refer Slide Time: 19:30)



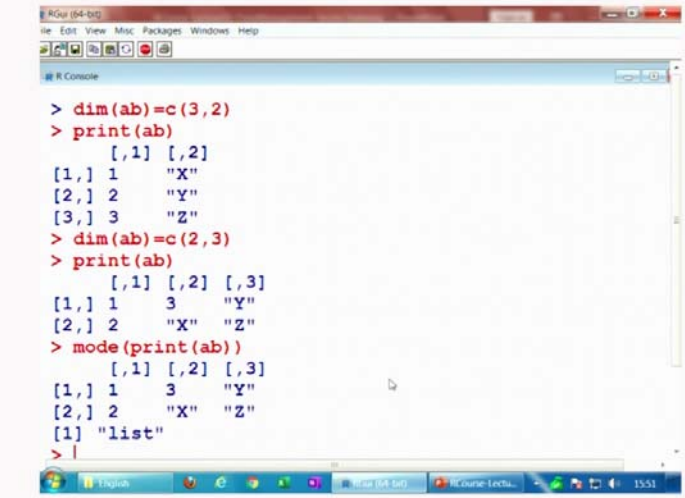
```
R Console  
> ab = list(1, 2, 3, "X", "Y", "Z")  
> ab
```

(Refer Slide Time: 19:32)



```
R Console  
> ab  
[[1]]  
[1] 1  
  
[[2]]  
[1] 2  
  
[[3]]  
[1] 3  
  
[[4]]  
[1] "X"  
  
[[5]]  
[1] "Y"  
  
[[6]]
```

(Refer Slide Time: 19:36)



```
> dim(ab)=c(3,2)
> print(ab)
  [,1] [,2]
[1,] 1  "X"
[2,] 2  "Y"
[3,] 3  "Z"
> dim(ab)=c(2,3)
> print(ab)
  [,1] [,2] [,3]
[1,] 1  3  "Y"
[2,] 2  "X" "Z"
> mode(print(ab))
  [,1] [,2] [,3]
[1,] 1  3  "Y"
[2,] 2  "X" "Z"
[1] "list"
> |
```

So, now you try to give here dimension of ab is like say c let us try to do give 3 comma 2, right and then if I try to see here print ab and see here this is like here 1 2 3 “X” “Y” “Z”. And if you try to give here see here 2 comma 3 then it will be like here this, right. So, you can see that the data is arranged column wise and if you try to see the mode of here this one you can see here this is here list, right.

So, now, we come to an end to this lecture and you can see here that we have considered very elementary operation in this lecture, my objective was that there are many many things which are very small and I want to compile all of them in this lecture. So, that means I can give a logical break to such operations, but definitely there is a very long list of such operations which can be done over the R software.

So, I will stop here with these operations, but my request to you all will be just try to look into the books try to look into different resources and try to see what are the various other operations which are available and particularly the type of operation which you want, because you are doing work in a particular area, in a particular field and you would like to continue in that. So, you will need those tools for those operations.

So, I am so, I have tried my best to cover here the most commonly used operation, but you will need to know such operation in the area in which you are working. So, you try to look for them, try to practice them and I will see you in the next lecture till then goodbye.