

Introduction to R Software
Prof. Shalabh
Department of Mathematics and Statistics
Indian Institute of Technology, Kanpur

Lecture – 20
Vector Indexing

Welcome to the next lecture on introduction to R software. In this lecture we are going to talk about some issues related to the indexing of a vector. We have seen that we can create a vector and there are certain entries inside the vector and every entry has an address. Now when I want to manipulate with the addresses of those entries, then how to do it; there can be certain situations in the programming where you can call a particular value through the index for example, suppose there are 20 entries in a vector and I want to choose only those entries in the vector for which the indices are say from 3 to 8 or I want to choose those types of entries for which the indices are in some even numbers.

For example in the data set of 20 values I want to choose the values which are at the second position, fourth position, sixth position, eighth position and so on. So, those values can be called using the index of the vector. These types of operations are needed whenever we are trying to do the programming. So, now, we are going to understand how we can play with the indexing of a vector.

(Refer Slide Time: 02:08)

Vector Indexing

A vector of positive integers (`letters` and `LETTERS` return the 26 lowercase and uppercase letters, respectively).

```
> letters[1:3]
[1] "a" "b" "c"

> letters[ c(2,4,6) ]
[1] "b" "d" "f"

> LETTERS[1:3]
[1] "A" "B" "C"

> LETTERS[ c(2,4,6) ]
[1] "B" "D" "F"
```

The slide includes two R Console screenshots. The first screenshot shows the command `> letters[1:3]` resulting in `[1] "a" "b" "c"`. The second screenshot shows the command `> letters[c(2,4,6)]` resulting in `[1] "b" "d" "f"`. Handwritten green annotations highlight the indices and the resulting output values in both the text and the screenshots.

2

First of all we try to look at the vector of the positive integers; that means, they are containing the value 1 2 3 4 and so on. One option to play with the vector of positive integers is to use the lower case or say upper case alphabets. In an earlier lecture we had discussed that how the lowercase alphabets and how the uppercase alphabets can be called in R programming and if you remember we had used the syntax see the small letters or a lowercase case l e double t e r s letters to call the lowercase alphabets like in a small a small b small c and so on and using the capital letters l e double t e r s that is the syntax, we can call the capital letter alphabets like as capital A capital B capital C capital D and we had also understood that these letters are chosen by their index.

For example, when I say the letter at index number one this means a, when I say the letter at index 2 this means b. So, this is one simple option where I can call the positive integers as an index using the alphabets. So, now let us try to take some example and try to understand that how we can operate, but you have to keep in mind that there are only 26 alphabets. So, depending on your need in case if you want to play with 26th numbers or say 20 sixth positions or 26 values you can use this a small letters or say capital letters alphabets right. So, let me take here some example, we have seen that when you try to use the syntax in letters you get a small letters and say capital letter alphabets in say small and in the say the lowercase.

Or say uppercase for example, just to have a quick review let us try to do it over the r console and you will see here.

(Refer Slide Time: 05:07)

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
[14] "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
>
> LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
[14] "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
>
> letters[1]
[1] "a"
>
> letters[14]
[1] "n"
> LETTERS[1]
[1] "A"
> LETTERS[14]
[1] "N"
> |
```

This is letters and similarly if you try to get your capital letters that can be obtained by typing l e double t e r s in uppercase. Now suppose I want to see what is there in the first position, I can say here letters at position number one and I get here a. Similarly if I write down here what is the letter at position number 14, I get here a small n and the same thing is possible when I try to do with the capital letters the first capital letter or the first alphabet in the syntax l e double e r s in uppercase it is a and.

Similarly, if I want to know what is the fourteenth letter then this is here n. Now what do you observe here? And if I ask you a simple question what is an index? Now you can easily understand what do what do we really mean by an index whatever I am writing here inside this bracket, this is an index and when I see these sequences of letters I can say here that first letter which I am highlighting here a this is in the first position, second letter b this is in the second position, third letter c this is in the third position and so on. So, now, you can see here in this command which I am highlighting here I am trying to call the first letter by writing here one and whatever I am writing here inside this bracket inside this square bracket that is the index.

So, we are now going to see here how to use this index to manipulate the different values. So, you can see here that first of all suppose I want to call 3 possible values say a b and c. So, one option is this, I can write down here the sequence one colon three; that means, I am trying to play with the first second and third value, and whatever I am

writing here inside this bracket, square bracket this is actually the index. So, when I say that the topic of this lecture is vector indexing; that means, I want to know how I can play with the index of the vector to get different types of data set and how to do different types of data manipulations using the index of a vector. So, you can see here now I am trying to play with this here 1 2 3 and as soon as I enter here I get here a b and c.

Similarly, in case if I want to call any particular value, say suppose I want to have here the values at position number 2, position number 4 and position number 6 here. So, I try to write down all these indices inside this combined command c, and then I use the syntax here letters inside this a square bracket. And then you can see here I am getting here this output b d and f this means b means b is at the second position, d is at the fourth position and f is at the sixth position and you can see here I have given here the screenshot also.

And the same thing also you can do with the uppercase alphabets just by writing the capital letters or small letters in uppercase, and you can see here the same example I have repeated here and if you want to do it here in the r console also we can see here for example, I can write down here say this letters say here c and suppose I want to have at the twelfth position and say this twentieth position and say 26th position. Suppose I want to have 3 positions; that mean, the addresses of the 3 values are twelfth twentieth and 26 and you can see here I get here 3 letters l t and z.

So, this is one possible way out to play with the index or the indices of the numbers to get a particular type of data set or to generate a particular type of data set which can be further used in different types of applications. Now you try to take some other examples and we try to understand different types of application through this example.

(Refer Slide Time: 11:12)

```

Vector Indexing
□ A logical vector
> x <- 1:10
>x
[1] 1 2 3 4 5 6 7 8 9 10
> x[ (x > 5) ]
[1] 6 7 8 9 10

> x[ (x%%2==0) ] #%% indicates x mod y
[1] 2 4 6 8 10 #values for which x mod 2 is 0

> x[ (x%%2==1) ]
[1] 1 3 5 7 9 #values for which x mod 2 is 1

```

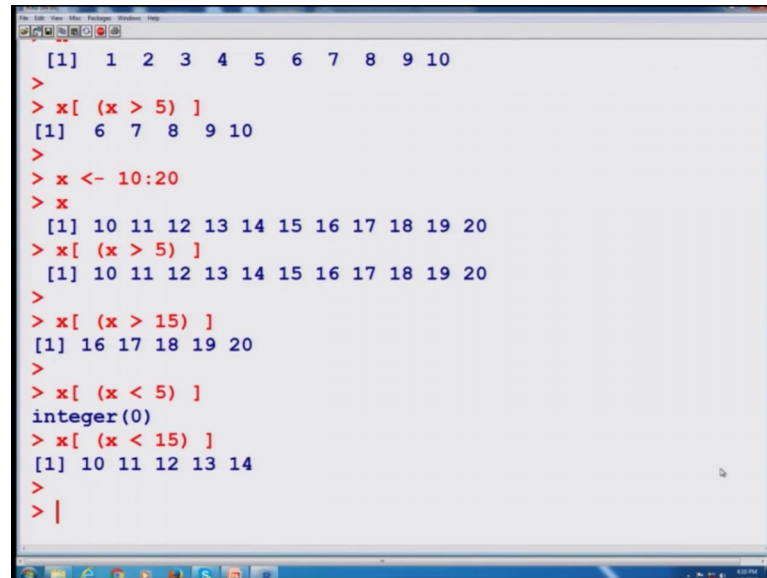
Now, instead of using the syntax letters, I am now using the concept of logical vector and using the concept of logical vector I can play with the indexing of a vector for example, suppose I try to generate here a sequence of numbers which are stored in the variable x as 1 to 10, 1 colon 10; that means, a sequence is starting from 1 to 10 at an interval of one units. So, if you try to get this sequence it will look like this. Now suppose I want to get here the values which are greater than here 5 from one basic knowledge we can see here one is not smaller than 5, 2 is not smaller than 5, 3 is not a smaller than 5, 4 is not a smaller than 5 and 5 is also not a smaller than 5.

But 6 7 8 9 10 they are greater than 5. So, I want to extract these values using the are programming. So, what I try to do here that first I try to denote these values as an index of a number right and now I want all the values which are greater than 5. Fortunately what is happening here that the number and the index they are the same here, but I have taken this example intentionally so that you can understand it clearly right. So, if you try to see here first of all if I try to write down here x greater than 5, this is my index and then I try to write down all the numbers in the in the vector x for with this logical vector is giving me an output for the numbers which are more than 5 which are greater than 5.

So, this is what I try to do here say I try to write down this part and you can see here this is my here logical part. So, I am trying to play with the index using the logical operator. So, as soon as I say here that I need all the values at the index which is greater than 5; obviously, the control comes at 6 7 8 9 10 and we get this output, but before that what is happening? There is another bracket here and before that there is a value here or a

variable here x. So, when I try to write down here this x, this is saying that please try to find out all the values assign in the x vector which are greater than 5, and this gives us an answer 6 7 8 9 and 10. So, let us try to do this thing in the r console.

(Refer Slide Time: 14:45)



```
[1] 1 2 3 4 5 6 7 8 9 10
>
> x[ (x > 5) ]
[1] 6 7 8 9 10
>
> x <- 10:20
> x
[1] 10 11 12 13 14 15 16 17 18 19 20
> x[ (x > 5) ]
[1] 10 11 12 13 14 15 16 17 18 19 20
>
> x[ (x > 15) ]
[1] 16 17 18 19 20
>
> x[ (x < 5) ]
integer(0)
> x[ (x < 15) ]
[1] 10 11 12 13 14
>
> |
```

So, I can write down here x 1 to here 10. So, you can see here now this x is a sequence of number from 1 to 10, now I try to write down here x greater than 5, you can see here that I am getting here 6 7 8 9 10 and suppose I take here another sequence see you from here x 10 to here 20 right suppose this is here you can see here that is the sequence starting from 10 at an interval of 1 then going from 10 to 20. Now suppose I try to repeat the same thing what do you get here it is trying to give me all the numbers which are greater than 5 yes the sequence is starting from 10.

So, all the numbers here are greater than 5 starting from 10 to 20, but now if I try to take here all the numbers greater than 15 then this is only 16 17 18 19 and 20 and suppose if I try to find out here all the numbers which are the smaller than 5 let us see what happened, there is no value because all the values are greater than 10. So, there is no value which is a smaller than 5, but if I say here I need all the values which are the smaller than 15 then we get here that is the 10 11 12 13 and 14. So, you can see here this is how we can play with this thing right.

Different types of such operations can also be done with the same vector, now let us try to take some more example. Suppose I try to take here the same vector here that x going

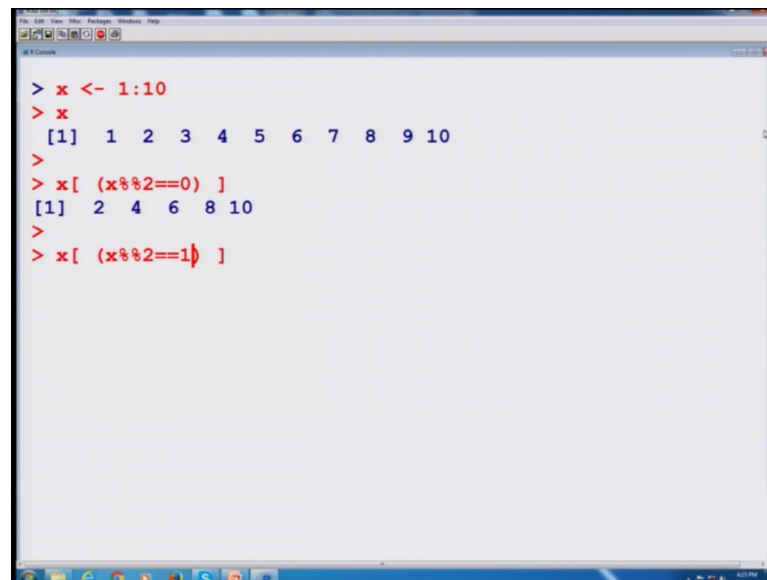
from one to here 10 and now in this sequence I would like to know that what are the values for which $x \text{ modulo } 2$ is 0 that is my question; that means, from the sequence from 1 2 3 up to here 10, I would like to find out the values from here for which $x \text{ modulo } 2$ is equal to 0 you may recall that we had discussed the operation of $x \text{ modulo } 2$ in initiate lectures. So, the same thing is being copied here on each and every element of the vector and we had also discussed that when you are trying to operate the $x \text{ modulo}$ operator in a vector then this $x \text{ modulo}$ operation carries over each and every element in the vector.

So, the same thing happens here also that all these values of x from 1 to 10 it is trying to operate whether $x \text{ modulo } 2$ is exactly equal to 0 and this operation is happening first at x equal to 1, x equal to 2, x equal to 3 and up to x equal to 10 and then it gives me all the values for which the $x \text{ modulo}$ comes out to be 2 for example, if I try to say here 2 divided by here 2 this is here the answer comes out to be here like this divisor is one and the remainder is 0 and similarly if I try to make it here 4 divided by 2 the remainder again comes out to be 0. So, 2 and 4 are the required values and similarly this operation goes over the other value 3 4 5 6 7 8 9 and wherever it gets the answer to be 0 it reports here.

Now, similarly when we are trying to do the $x \text{ modulo } 2$ over a sequence of 1 to 10 then only 2 types of remainders are possible either 0 or say 1. So, let us try to see what happens when I try to make my condition that, that please try to give me all the number for which $x \text{ modulo } 2$ is exactly equal to 1. Please remember one thing this sign to equality sign; that means, exactly equal to and this was a logical operator so; obviously, if you try to divide 3 by 2 you will get here a remainder one similarly if you try to divide 7 by 2 you will get a remainder one all these values 1 3 5 7 9 they are indicated here .

Let us try to do the same thing in the r console and try to see what do we get over here so. Firstly, let me try to generate here the.

(Refer Slide Time: 20:09)



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

Values what we are trying to use here you can see I can just scroll the upward keys and can come back to an earlier command this is my here x and from here I try to find out the values for which x modulo 2 is equal to 0 and this answer comes out to be here 2 4 6 8 10 and similarly when I try to do it for here 1; that means, x modulo 2 is equal to 1; that means, all the numbers in which the remainder is coming out to be one when it even when they are divided by 2 these answers are 1 3 5 7 9 right note.

Let us come back to our slides and here is the screenshot of this operation. So, you can just try it yourself also.

(Refer Slide Time: 21:13)

```
Vector Indexing  
□ A logical vector x = 1, 2, ..., 10  
> 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
```

Handwritten notes:
- A green arrow points from `x[5]` to the `NA` value in the output.
- A green oval encircles `!is.na(x)` with a note "Means negation".
- A green arrow points from the `!` to the text "Means negation".
- A green arrow points from the `5` in `x[5]` to the `5` in `5 → NA removed`.
- A green arrow points from the `5` in `5 → NA removed` to the `5` in `5 is missing`.
- A green arrow points from the `5` in `5 is missing` to the `5` in `5.555556`.
- A green arrow points from the `5` in `5.555556` to the `5` in `5 → NA removed`.
- A green arrow points from the `5` in `5 → NA removed` to the `5` in `5 is missing`.

Now let us try to take another example and I try to consider here the same sequence from 1 2 up to here 10 and suppose I replace the fifth value by NA not available. Now you have to observe one thing my objective was to replace only the value at the fifth position how I am calling it. This I am trying to address by here index the index here is 5 and using this index I am trying to call the value at the fifth position and by this command that x bracket 5 is equal to NA I am trying to say please replace whatever is the value at fifth position by NA. So, now, you can see here this output comes over here and.

Now, there is a value here NA now; obviously, we had seen earlier also and here I would like to show you that suppose if you want to find out the automatic mean of all the values inside the vector x, automatic mean is sum of all the observation divided by the number of observation. So, whenever you try to make it here 1 plus 2 plus 3 plus 4 up to here 10 divided by here 10 what do we expect? Here is an answer new mathematical operation can be done over this and it will say that the mean of x is say NA not available now think what do you want.

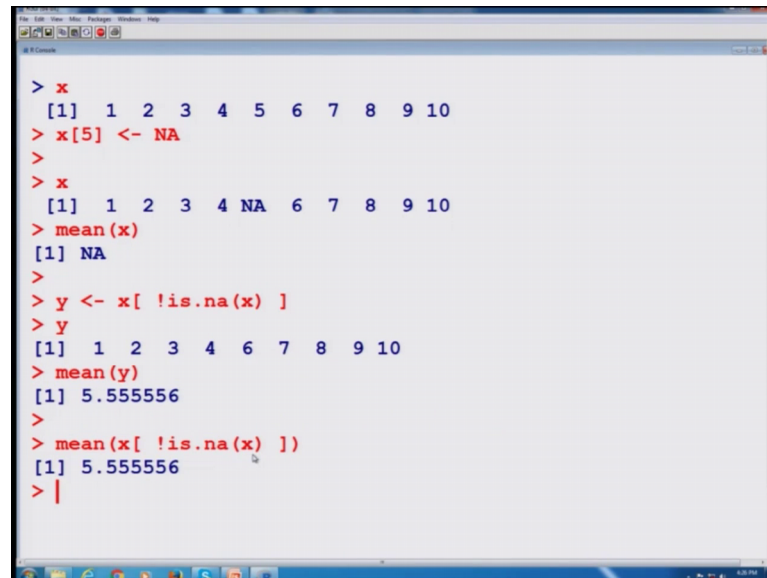
Suppose in your vector there are certain values which are missing that goes this goes without saying that what you really want is the mean of those values which are available as an integer or some number, you are simply asking that please remove the not available values and please give me the average or the mean of those values which are only available.

But definitely computer cannot understand what you are thinking, you have to instruct to your computer so; that means, I have to ask my computer that please consider the values inside the vector x, and please check or please remove the values which are not available and then find out the mean of the values which are available. Now how to do this thing inside the r, this can be done using the logical operator inside the index let us try to see how. So, I try to define here a new vector just for the sake of simplicity, once you get some practice you can do it in a single line actually. First I try to define here a vector in which the missing values are removed means if you remember.

We had used this exclamation sign when we discuss the logical vector that this exclamation sign means negation that means, I do not want. So, here I am trying to use a command here is there any NA this is again one of the command that we had used earlier. So, now, I am trying to find out that is there any NA inside this vector x and then I am saying that please remove them. So, I need here the value of here x after removing those NAs, and this command is given inside this bracket sign. So, I am trying to say in more simpler words please use this vector x and check if there is any not available values then I am trying to use exclamation sign and am I asking that I do not want this NA values, and with this bracket sign square bracket I am trying to say please give me only those values where there is no NA.

Now, I get here a vector here y in which you can observe that something is missing here in the original vector it was here 5. Now what we did we have replaced this 5 here by NA in the first step that we did here, and then from the sequence I have removed this NA and this we have done here and now I am simply trying to find out the mean of all the values which are contained in the y vector and this comes out to be like this 5.55. So, you can see here just by using the r just by manipulating the index using a logical operator we can obtain different types of outcome, let us try to do this operation on the r console and see what happens.

(Refer Slide Time: 27:10)



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

So, if you try to see here we already had defined here x , a sequence from one to 10. Now I am saying that the fifth value or the value at the fifth position should be replaced by see here NA and now you can see here what is my here x now there is n value here NA here is earlier it was here 5. Now if I try to find out the mean of x you what you get here no it is not available that we have seen y . Now I try to define here another vector in which I am trying to remove these NA values. Now if we use try to see here y values which is here something like this, 5 is now missing or NA is missing. Now if you try to find out here mean of your y you get this thing well in case if you have a some practice here then n then all these operations can be written in a single line for example, I can find out here the mean of all those values which are missing like this you can get here the same value you can see here this is the same operation, but just for the sake of understanding I am trying to elaborate it in say smaller step.

So, that you can understand it once you get practice you can combine these several operation in a single step. So, now, in the next slide I have given the screenshot of the same operation you can see here you can try here you can try it yourself now I take another example. Suppose I want to generate a sequence of certain numbers or suppose I have a sequence and from that sequence I want to extract a subsequence. So, let we take an example here to understand it that how this can be done just by manipulating the index of a vector.

(Refer Slide Time: 29:47)

Vector Indexing

□ Vector of negative integers

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

subsequence

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

has the same outcome as

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

$x[(1:5)]$
1, 2, 3, 4, 5
 $x[-(1:5)]$
6, 7, 8, 9, 10

7

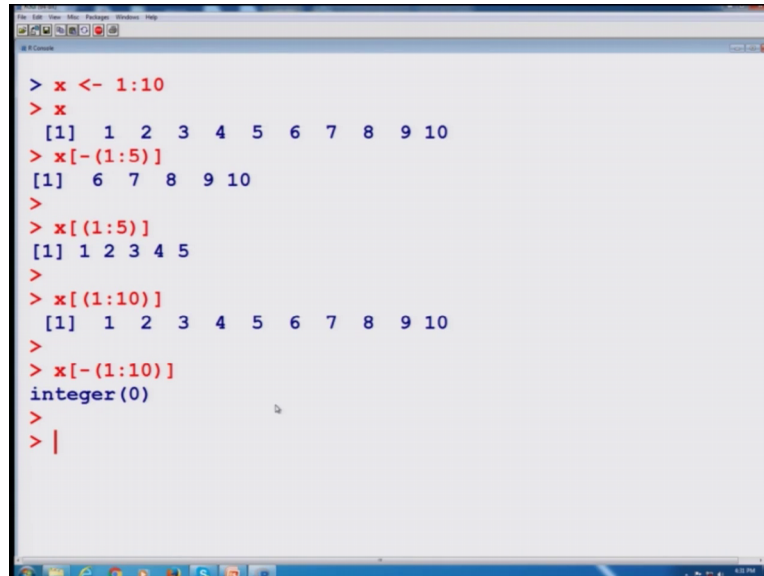
Suppose I try to take here the same example that I am trying to generate a data from one to 10. So, this is here the values and suppose I want to generate a sequence from say 6 to 10; that means, this is a sort of sub sequence. So, I have 2 options here more simple option is this that we already have done it is something like this I can write down here say here the variable name here x then inside this bracket say from 6 to 10 using this colon sign and you will get the output 6, 7, 8, 9, 10 what I want to show you here that this can also be done in an alternative way and.

Instead of writing here 6 to 10 like this one I write here like this. I try to write down the complement of 6 to 10. So, I try to write down here the remaining part that is here 1 2 5 inside this bracket I am trying to write 1 colon 5; that means, a sequence from 1 to 5 and then I try to put here a negative sign and you see what do we get here 6 7 8 9 10. So, now, if you try to see what you have really learned from this operation. In case if you try to put some sign like as positive sign or say negative sign inside the index then the operation is changed that is affected for example, if I want to write down here only the values of x from 1 to 5 I will show you that you will get here an answer 1 2 3 4 and 5, but as soon as you put here a negative sign.

Then you get here an answer 6 7 8 9 and 10. So, this negative sign is trying to change the ordering just complement just opposite. So, this also gives you another option to play with the index to get a required data set from a sequence let us try to do it here and try to

see what we do we what do we get here. So firstly, let us try to generate a sequence here x.

(Refer Slide Time: 32:43)



```
> 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[(1:5)]
[1] 1 2 3 4 5
>
> x[(1:10)]
[1] 1 2 3 4 5 6 7 8 9 10
>
> x[-(1:10)]
integer(0)
>
> |
```

So, you can see here these are the value from 1 to 10 and suppose I try to write down here this thing x 1 to 5 with the negative sign then I get here 6 7 8 9 10 and if I try to write down here 1 2 5 then I get here 1 2 3 4 5. So, negative sign is just trying to change the ordering it start instead of it starting from 1 to 10 that is the starting from 10 to 1 for example, another way to write down this sequence here you say the entire sequence x is 1 to 10 like this.

But if I try to write down here with this negative sign you can see here this is trying to say something else. What is really happening you expected that it will give you the sequence 10 to 1 it does not happen. So, you have to be careful when you are trying to play with the index of a vector well this is not possible for me to explore here all possible situations, but my very sincere advice to all of you is that whenever you are trying to do anything, please do not depend on your intuition. Whenever you want to do something please try to take a simple example and try to see what is the really computer is trying to do because computer cannot understand or cannot read our mind computer has a predefined language, but this is only who want a or who wants an output in the required format.

So, we have to follow that what really computer understands, but this is the advice which is valid for the entire programming throughout the life. So, now, let us try to a come back to our own situation and let us try to see what we have done here and this is the a screenshot of whatever we have done here. So, now, we stop here and here I have tried to explain you that there is an option to play with the index of a vector and based on that you can generate different type of data set, well way to use and how to use that depends on your logic.

Whenever you are trying to do r programming; that means, you are trying to instruct the computer to do something computer has its own language and we have our own language. So, we try to develop our logic based on that the computer will follow our instruction. So, whenever you need any type of data set, you have to think that what is the most efficient way to generate the data, that can be through the data vector itself or by playing with the index of the data vector. So, you practice it enjoy this and we will see you in the next lecture till then good bye.