Foundations of R Software

Prof. Shalabh
Department of Mathematics and Statistics
Indian Institute of Technology, Kanpur

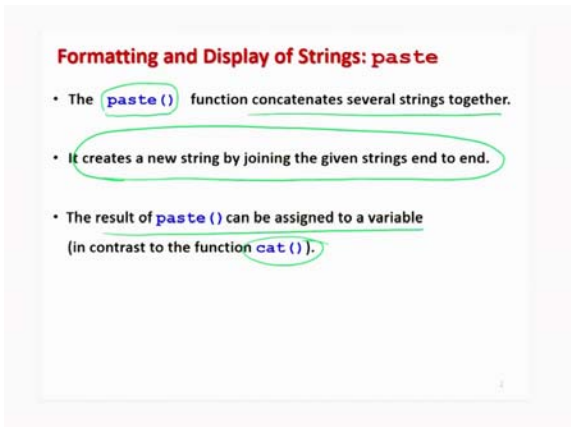
Strings - Display and Formatting
Lecture - 37
Paste Function

Hello friends. Welcome to the course Foundations of R Software. You can recall that in the last two lectures we initiated a discussion on that how to control the output in the R software. And you wanted to have the output in a particular pre specified format. Format does not mean format that we have used in the print and cat, but it is a general format, right.

So, now, we have learnt that how to print the output using the print function and cat function. Now, in this lecture today we are going to talk about a new function, which is about paste, right. We already have used the paste couple of times in the earlier lectures, but this is the lecture where we are going to formally learn what is the role of paste and what it exactly does.

So, as the paste means something like pasting, joining together, etcetera. So, with a very simple and literal meaning of the paste you can understand what paste is going to do. But here I am going to take a couple of options with the paste and try to show you that how the paste can be used in different ways to get different types of outcome. So, let us begin the lecture and I will try to take up here couple of examples to explain you.

(Refer Slide Time: 01:33)



Formatting and Display of Strings: paste

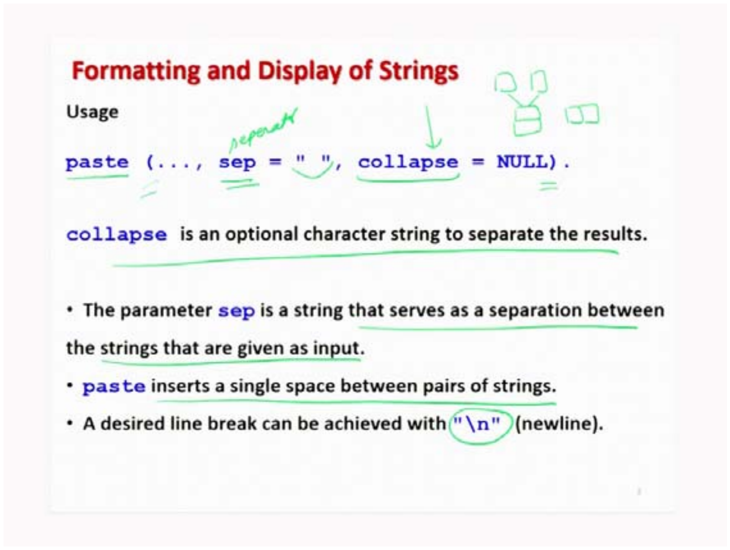
- The `paste()` function concatenates several strings together.
- It creates a new string by joining the given strings end to end.
- The result of `paste()` can be assigned to a variable (in contrast to the function `cat()`).

Now, if you try to see here the function for pasting is paste; paste and then inside the bracket within the parenthesis within the arguments we try to give different types of option. So, the role of paste function is to concatenates several strings together, right. And what it does? The cat also does the same thing that it concatenates several things several strings together and along with the numbers, right.

What really happens in the case of paste that it creates a new string by joining the given strings end to end and the advantage is that the outcome of the paste can be assigned to a variable, which is not possible in the case of cat. So, although cat is also doing the concatenation, but the outcome of the cat function cannot be assigned to a variable which can be used further, but in the case of paste this is possible.

So, that is the main difference between the cat and paste in spite of they are doing the same job, but they are different.

(Refer Slide Time: 02:28)



Formatting and Display of Strings

Usage

`paste (..., sep = " ", collapse = NULL).`

`collapse` is an optional character string to separate the results.

- The parameter `sep` is a string that serves as a separation between the strings that are given as input.
- `paste` inserts a single space between pairs of strings.
- A desired line break can be achieved with `"\n"` (newline).

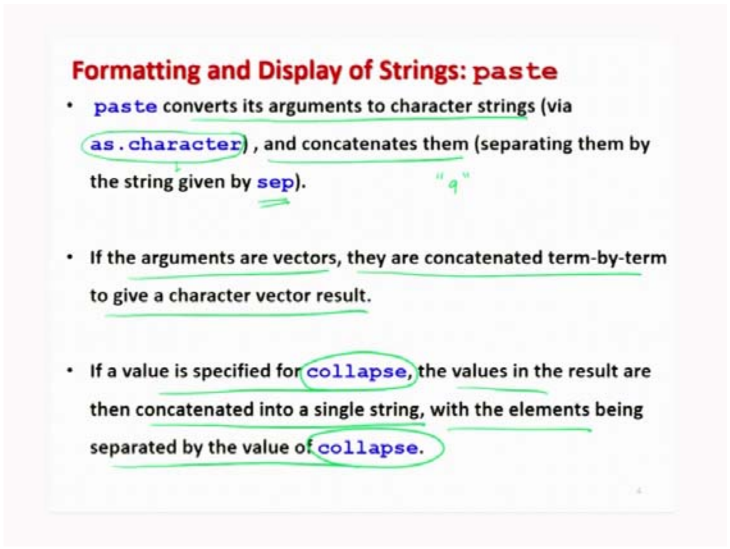
Now, if you want to use the function here paste. So, we simply write down here p a s t e and then inside the parenthesis we try to write down the different options and there are many more options. And I would request you to go to the help and then try to see what are the different options which can be used. But here I am trying to basically explain you about two important options, one is here the sep, that you know this means separator.

So, this parameter separator that helps in separating different strings and this can be controlled, in the sense that whatever separator you want to give you can give inside the double quotes. So, this is the same thing what we have done earlier also. Now, there is another option here which is collapse c o l l a p s e all written in the lowercase alphabets. So, what do you understand with the meaning of this word collapse?

So, collapse means if there are two things like as here, when they collapse then they are over each other or they are the side by side whatever you want. So, they are trying to stick together. So, this collapse is an optional character string to separate the results, right. So, what does actually paste does?

That if you do not give here any option for the separator then paste will try to insert a single space between the pairs of a string and after this we would like to break up the line using the command, backslash n that we have used in the earlier lectures also. So, this will give the outcome in the new line next line, ok.

(Refer Slide Time: 03:55)



Formatting and Display of Strings: `paste`

- `paste` converts its arguments to character strings (via `as.character`), and concatenates them (separating them by the string given by `sep`).
- If the arguments are vectors, they are concatenated term-by-term to give a character vector result.
- If a value is specified for `collapse`, the values in the result are then concatenated into a single string, with the elements being separated by the value of `collapse`.

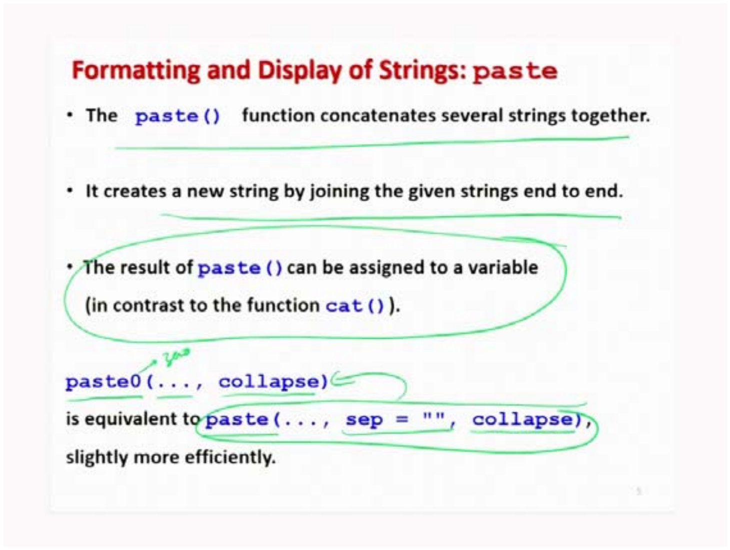
So, now the next question comes, how paste actually work? This is very important for you to understand so that you can compare it with different types of other commands like cat. Do you remember that we had done a command as dot character that it will try to convert for example, if I try to take a number as here 9 and I try to use here as dot character.

And inside the parenthesis, I write the 9 then it will try to give me an output 9, right. So, now, this paste works based on the as dot character. So, what it does that paste is going to convert whatever the values are given in its argument. That is within the parenthesis using the command as dot character and then concatenates them and if you want to separate them then it will use the option form sep otherwise it will give a blank space.

So, essentially whatever you try to give within the parenthesis of the paste function, they all are automatically converted to character and then they are concatenated together, they are joined together they are linked together exactly in the same sequence. So, in case if the arguments are vectors, then they are concatenated term by term to give a character vector result. That means, if you are trying to take more than one this vectors then they are also concatenated term by term.

And finally, the outcome is going to be a single character strain. And in case if a value for the option collapse is defined that is specified, then the values in the results are concatenated into a single string with the elements being separated by the value of collapse. Whatever value of collapse that you have given that will be inserted and then they will be collapsed together. Well these things will be very clear when I try to take the examples, right.

(Refer Slide Time: 05:37)



Formatting and Display of Strings: `paste`

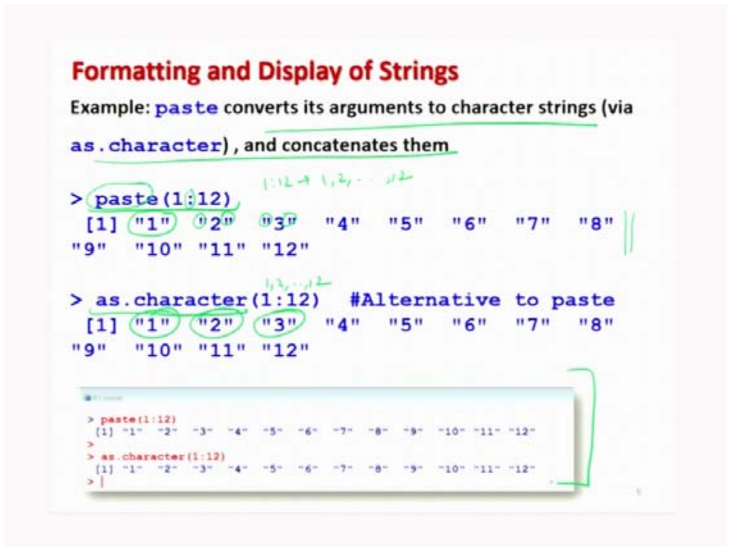
- The `paste()` function concatenates several strings together.
- It creates a new string by joining the given strings end to end.
- The result of `paste()` can be assigned to a variable (in contrast to the function `cat()`).

`paste0(..., collapse)` is equivalent to `paste(..., sep = "", collapse)`, slightly more efficiently.

So, now in a nutshell we have learned that the paste function concatenates several strings together and it tries to create a new string by joining the given string end to end. And the result of the paste can be assigned to a variable which is not possible in the function cat. And beside this paste function there is another function here paste0. So, this is here you can see here this is 0 not o that is the number 0.

So, paste0 is equivalent to this option that paste where you are trying to use separator where there is no space and then you are trying to use here collapse. So, if you want to do such an operation, then instead of using this thing you can simply use here the option paste0, whatever you want to give here and then simply use the option here collapse ok. This thing will be clear when I try to take an example, I will try to show you that when you try to use the paste function and the paste 0 function, how the things work, but anyway.

(Refer Slide Time: 06:31)



Formatting and Display of Strings

Example: paste converts its arguments to character strings (via as.character), and concatenates them

$1{:}12 \to 1, 2, \ldots, 12$

```
> paste(1:12)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"
"9"  "10" "11" "12"
```

$1, 2, \ldots, 12$

```
> as.character(1:12)  #Alternative to paste
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"
"9"  "10" "11" "12"
```

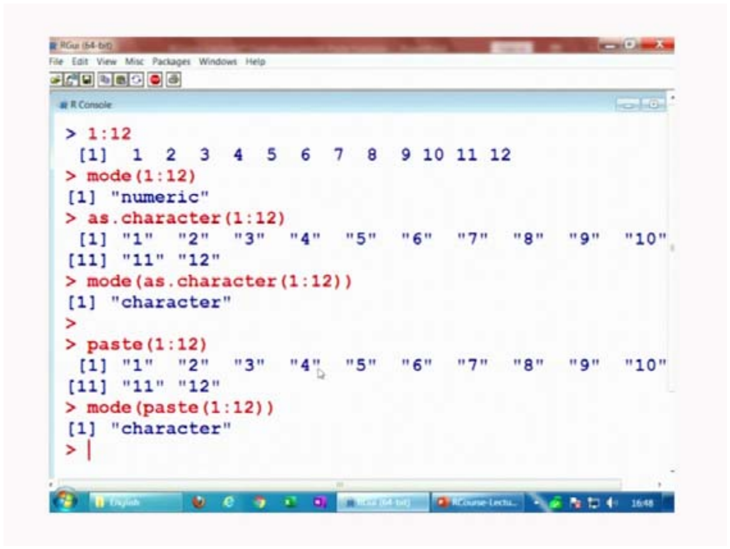
```
> paste(1:12)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
>
> as.character(1:12)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
> |
```

Now, let us start taking some examples, so that I can justify whatever I have explained you. So, as I said how the paste works. So, paste converts it is a argument to a character string via the as dot character and then concatenates them. That means, it links them together in the same sequence. So, now, let me try to give you here two things and then you try to compare. Suppose I try to use here the function paste 1 colon 12. So, they are going to be the numbers 1 colon 12, is 1, 2, 3, 4 up to here 12.

So, now you see these are numbers, but when you are trying to do it here paste, then the outcome is going to look like this. And if you can see here that every number is a now a character, this is enclosed within the double quotes. And in case if you try to take only the numbers 1 to 12 and if you try to use the command here as dot character; that means, whatever are the number here 1, 2, 3, 4 up to 12, now they are converted into a character and then you will see the same outcome here, right.

That all the numbers are enclosed within the double quotes, now they have become character. So, you can see here this is how the paste command work, right. And this is here the screenshot of the same outcome. So firstly, let me try to show you this example and then you will learn it enough, at least you are clear about this concept how it works, then you will understand it very clearly.

(Refer Slide Time: 07:50)

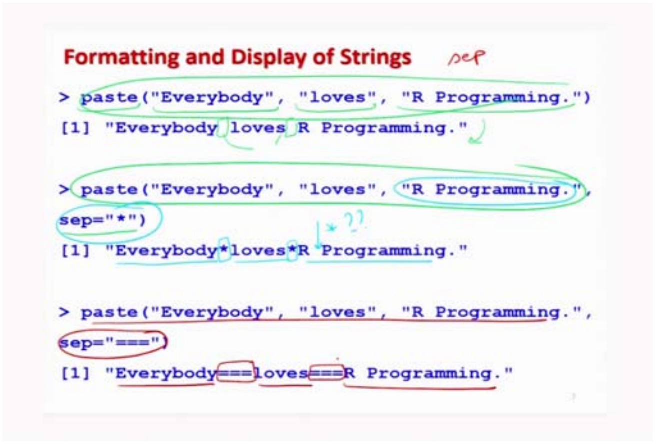


```
> 1:12
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> mode(1:12)
[1] "numeric"
> as.character(1:12)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
[11] "11" "12"
> mode(as.character(1:12))
[1] "character"
>
> paste(1:12)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
[11] "11" "12"
> mode(paste(1:12))
[1] "character"
> |
```

So, if you try to see here if I try to write down here the numbers this is here like this and if you try to see here the numbers, the 12 have the mode, which is here numeric. But in case if you try to use here as dot character and if you write down here 1 colon 12, then you can see here what it happens it is like here like this. And now if you try to see its mode this comes out to be here as a character.

So, you can see here this is happening and now if you try to do the same thing you simply try to write down a paste and simply write 1 to 12, you can see here this is here like this. And if you try to find out the mode of this here paste you can see here this is again character. So, this is how paste work this is exactly what I want to explain you, right, ok let me try to take here some more example and try to show you that how it works with different options.

(Refer Slide Time: 08:40)



Formatting and Display of Strings

sep

```
> paste("Everybody", "loves", "R Programming.")
[1] "Everybody loves R Programming."

> paste("Everybody", "loves", "R Programming.",
sep="*")
[1] "Everybody*loves*R Programming."

> paste("Everybody", "loves", "R Programming.",
sep="===")
[1] "Everybody===loves===R Programming."
```

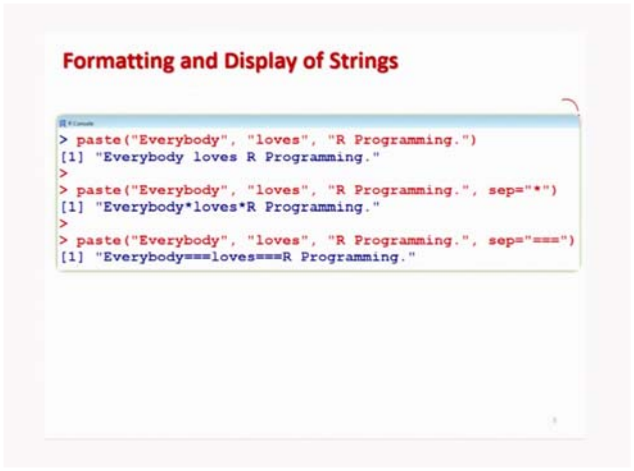
So, now I would like to give you an example of the option sep that is separator. So, I will try to take here an example, a statement and then I will try to use the separator and then try to compare what really happens. So, if you try to see I am trying to take here 3 characters, “everybody”, “loves”, “R programming”, right and if you try to paste them together they will be here like this.

So, you can see here every this character has a spacing, which is here single space, that is the default that is what I explained you in the beginning. Now, I try to use the same statement, you can see here from here to here I am writing here the same thing, but I try to add here sep is equal to say star and I try to write it in the within double quotes. So, as soon as I try to write down here this separator here you can see here the difference, still I have everybody loves R programming.

But the space between them it is separated by say star, do not get confused that why there is no star here, because you are trying to take here R programming as a single character. So, that is why it is within the double quotes and that is why there is no star between R and programming.

Now, similarly if you want to take here some other separator instead of a star, just try to take the same statement and try to use here the sep is equal to say this 3 times equal to sign, right. So, it is just a symbol, so you can see here now we have here everybody loves R Programming and then there are separators which are trying to separate the different strings.

(Refer Slide Time: 10:19)

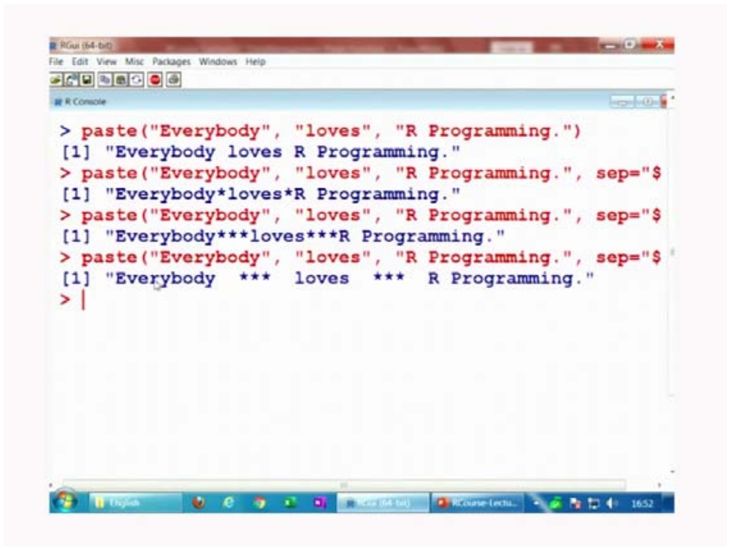


Formatting and Display of Strings

```
> paste("Everybody", "loves", "R Programming.")
[1] "Everybody loves R Programming."
>
> paste("Everybody", "loves", "R Programming.", sep="*")
[1] "Everybody*loves*R Programming."
>
> paste("Everybody", "loves", "R Programming.", sep="===")
[1] "Everybody===loves===R Programming."
```

So, you can see here this is not a very difficult command to understand and this is here the screenshot of the same operation, right. So, let me try to first show you this thing on the R console and then we will try to do some more operations, right.

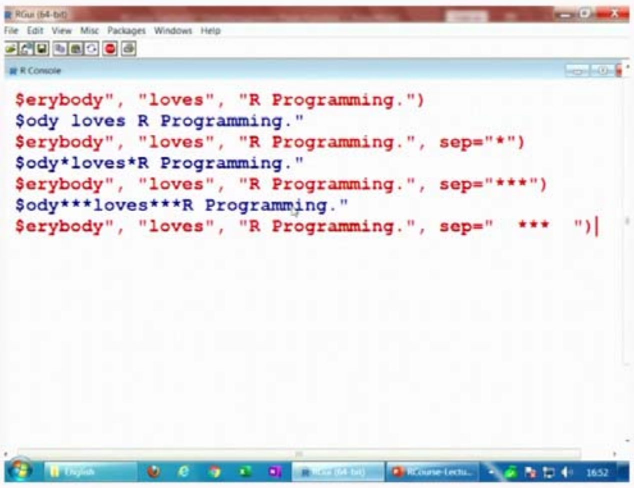
(Refer Slide Time: 10:33)



```
> paste("Everybody", "loves", "R Programming.")
[1] "Everybody loves R Programming."
> paste("Everybody", "loves", "R Programming.", sep="$
[1] "Everybody*loves*R Programming."
> paste("Everybody", "loves", "R Programming.", sep="$
[1] "Everybody***loves***R Programming."
> paste("Everybody", "loves", "R Programming.", sep="$
[1] "Everybody  ***  loves  ***  R Programming."
> |
```

So, you can see here I just try to first take the first statement here that everybody loves R programming, there is no separator and then I try to add here the separator. Separator is equal to say here within double quotes, which is star, right and you can see here that this comes out to be here like this. Everybody loves, now there is no spacing between here R and programming because this is here like as a single string and a separator here is like this, right.

(Refer Slide Time: 11:02)

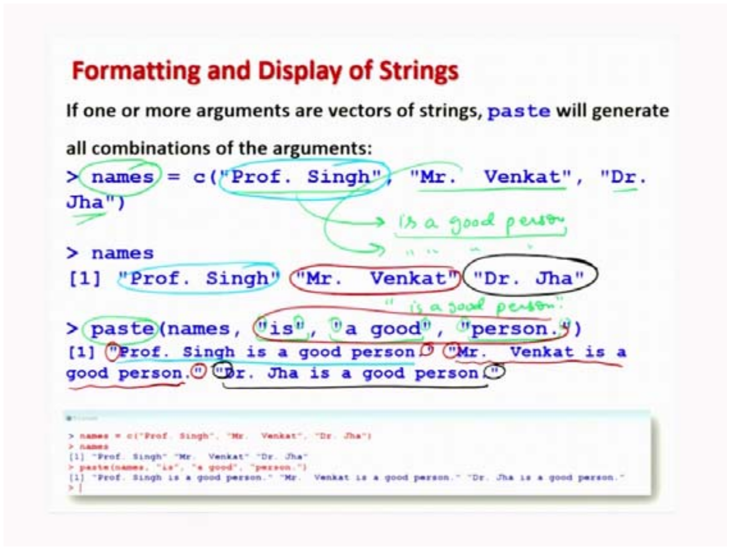


```
$erybody", "loves", "R Programming.")
$ody loves R Programming."
$erybody", "loves", "R Programming.", sep="*")
$ody*loves*R Programming."
$erybody", "loves", "R Programming.", sep="***")
$ody***loves***R Programming."
$erybody", "loves", "R Programming.", sep="  ***  ")
```

And similarly, if you try to take here means another here means another separator. Suppose I try to take it here see here three stars, you can see here this becomes a like this everybody loves R Programming and the separator becomes here like three stars and if you want to write down here something more.

If you want to write down here two space and two space after, the stars in the earlier command you can see here it becomes a like this. So, space is also taken as a separator. So, this is how this command actually works. So, I am sure that now it is not difficult for you to understand that how this paste command is working.

(Refer Slide Time: 11:34)



Formatting and Display of Strings

If one or more arguments are vectors of strings, paste will generate all combinations of the arguments:

```
> names = c("Prof. Singh", "Mr.  Venkat", "Dr.
Jha")

> names
[1] "Prof. Singh" "Mr.  Venkat" "Dr. Jha"

> paste(names, "is", "a good", "person.")
[1] "Prof. Singh is a good person." "Mr.  Venkat is a
good person." "Dr. Jha is a good person."
```

```
> names = c("Prof. Singh", "Mr.  Venkat", "Dr. Jha")
> names
[1] "Prof. Singh" "Mr.  Venkat" "Dr. Jha"
> paste(names, "is", "a good", "person.")
[1] "Prof. Singh is a good person." "Mr.  Venkat is a good person." "Dr. Jha is a good person."
> |
```

Now, I try to give you one more application, right. Suppose you have a condition here like this, that you want to consider a vector of the characters and it has got some couple of names. For example, you can see here I am trying to write down here three names, Professor Singh, Mr. Venkat and Dr. Jha, right.

And after that I want to write whatever is the name, after that I want to write is a good person and then I want to repeat it, yeah I do not want to use here the concept of loop. So, I want to write down here whatever is the name here Professor Singh, here and it is printed at Professor Singh is a good person, then the next name Mr. Venkat is taken and then it is printed here as say is a good person and the same thing is repeated with the Dr. Jha which is the third name.

So, now this thing can be done in the R software very easily. Well I am trying to take it at a very elementary level otherwise this is only a one line command. So, what I try to do here, first I try to store all the names in a data vector, say here names n a m e s, right and yeah then I try to use here the command here paste.

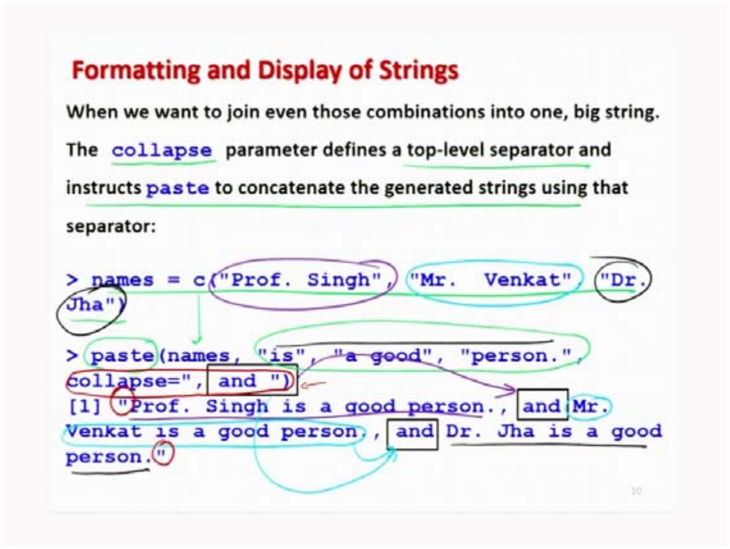
And then I try to write down here is a good person and full stop. And I try to write them inside the double quotes means if you wish you can also write them in a single quotes also like as is a good person, that also can be done, but anyway I am trying to explain you in a very lucid way.

So, now what will happen? That R will start working on it and it will try to pick up the first name from here Professor Singh or say here and it will try to write or combine it with all other strings which is here Professor Singh “is” “a good” “person” and after that it will come to the next name in the vector names and it will pick up here, Mr. Venkat and once again it will try to operate it with is a good person. So, it will print here Mr. Venkat is a good person.

And you can see here now this Professor Singh is a good person is a single string and similarly Mr. Venkat is a good person is a single string. And similarly then at the end it will try to pick up the third name which is here Dr. Jha and it will try to add here is a good person. So, it is trying to combine here Dr. Jha, then is then a good and then person.

So, these are 4 strings, but they are combined together and you get here an outcome, Dr. Jha is a good person and this is now single string. So, that is the advantage actually in the paste command and now you can see as you are moving further you can simply use these things in a different way.

(Refer Slide Time: 14:08)



Formatting and Display of Strings

When we want to join even those combinations into one, big string. The `collapse` parameter defines a top-level separator and instructs `paste` to concatenate the generated strings using that separator:

```
> names = c("Prof. Singh", "Mr.  Venkat", "Dr.
Jha")

> paste(names, "is", "a good", "person.",
collapse=", and ")
[1] "Prof. Singh is a good person., and Mr.
Venkat is a good person., and Dr. Jha is a good
person."
```

And now I try to show you here the application of collapse also and then I will try to show it on the R console. So, this option here collapse c o l l a p s e this is a parameter that defines a top level separator and instruct the paste command to concatenate the generated string using that separator, right.

For example, I will take here the same example and I will try to operate here with the collapse. You can see here I try to take here the same data vector which is has 3 values Professor Singh Mr. Venkat and Dr. Jha, right. And then I try to use here the option here paste and then I try to pick up the names from here and then I try to use here the same string is and another string are good and that the last string as person, right.

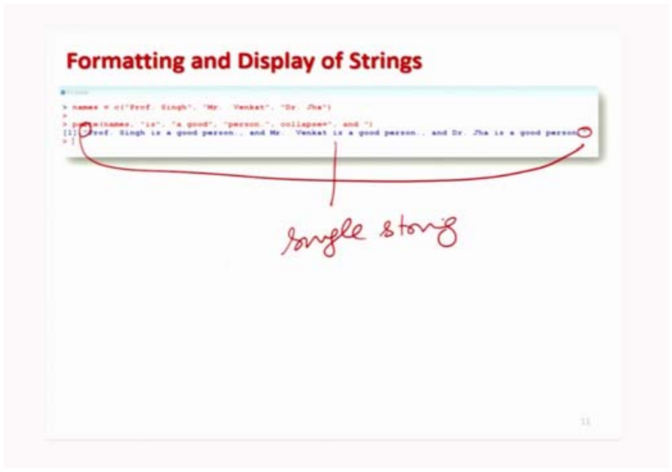
But now I add here collapse is equal to and. I have given here a black box just to indicate that how this and is appearing in the outcome, there is no other reason. Now, if you try to see here this outcome, how the outcome is working. So, it tries to pick up the first name here from here names, this is a Professor Singh and then it tries to combine it with Professor Singh is a good person full stop and after this it tries to add here and this is this and is added.

Here now in the second step, it will try to pick up the second name Mr. Venkat and it will try to add here is a good person and it will try to print here Mr. Venkat is a good person. But after that, now it will add here this and this and here is again added from this and which was given under the collapse and after that it tries to pick up the third name Dr. Jha and again it tries to use the is a good person and it tries to add here Dr. Jha is a good person.

Now, couple of things which you have to observe. First of all try to see whether this outcome is a single string or they are the three strings. For example, if you try to see in the earlier commands you had here 1st string here like this, Professor Singh is a good person, 2nd string Mr. Venkat is a good person and 3rd string was Dr. Jha is a good person.

You can see here 1, 2 and here 3, but when you are trying to work here with this collapse, it is giving you here only one double quotes here and here and all the three statements have been joined together. But the collapse which is joining the two statements is controlled by this double quotes and here backslash and, right. Anyway double quotes have to be there because it is a connective it is essentially the comma and this comma and they are appearing together here, right.

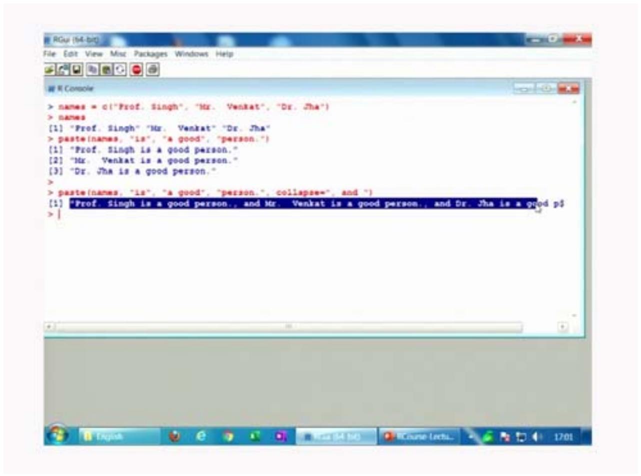
(Refer Slide Time: 16:46)


Formatting and Display of Strings

```
> names = c("Prof. Singh", "Mr.  Venkat", "Dr. Jha")
>
> paste(names, "is", "a good", "person.", collapse=", and ")
[1] "Prof. Singh is a good person., and Mr.  Venkat is a good person., and Dr. Jha is a good person."
> |
```

So, this is the use of this collapse and you can see here now here very clearly, there is here only these two double quotes are there. So, this whole is a single string. So, this is the difference between the separator and collapse. So, let me try to show you here these examples first on the R console and then I will try to move further, right. So, first let me try to create here this names here data vector which is going to be used in both the examples. So, names here is like this.

(Refer Slide Time: 17:14)



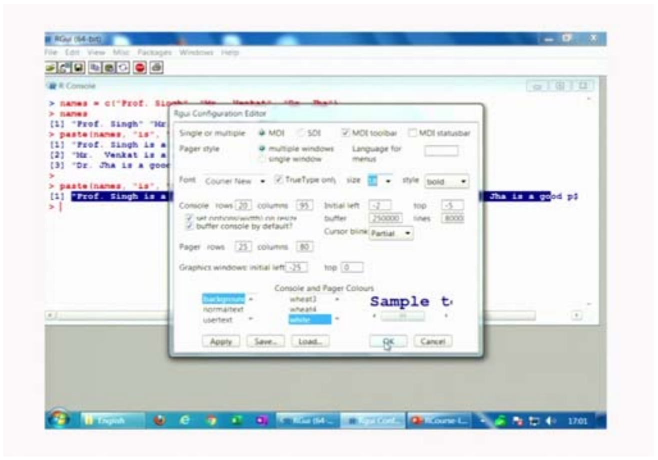
```
> names = c("Prof. Singh", "Mr.  Venkat", "Dr. Jha")
> names
[1] "Prof. Singh" "Mr.  Venkat" "Dr. Jha"
> paste(names, "is", "a good", "person.")
[1] "Prof. Singh is a good person."
[2] "Mr.  Venkat is a good person."
[3] "Dr. Jha is a good person."
>
> paste(names, "is", "a good", "person.", collapse=", and ")
[1] "Prof. Singh is a good person., and Mr.  Venkat is a good person., and Dr. Jha is a good p$
> |
```

So, you can see here there are three names here Professor Singh, Mr. Venkat and Dr. Jha. And then I will try to use here the command here for paste and in which I will simply trying to pick up the names from these data vector and then it will going to add is a good persons. You can see here professor sigh is a good person, Mr. Venkat is a good person, Dr. Jha is a good person this is just added from here, right.

And the names are coming from here Professor Singh, Mr. Venkat and Dr. Jha ok. So, now, after this I will try to take this here one more example where I try to use the paste command, but I use to, but I am using here the option for these here collapse. So, we can see here now these things are together yeah in order to see it I will have to reduce the phone size otherwise because this is a long statement, so you cannot see.

But you can see here this is a single statement, you can try it on your computer also and I will bring my this phone size back otherwise you cannot see it, anyway.

(Refer Slide Time: 18:10)

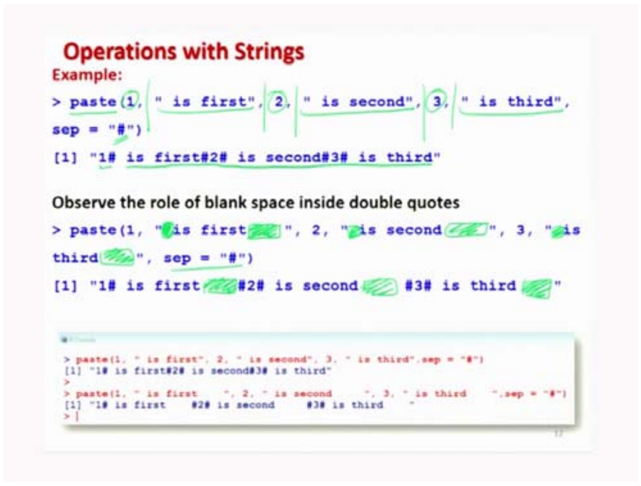


```
> names = c("Prof. Si
> names
[1] "Prof. Singh" "Mr
> paste(names, "is",
[1] "Prof. Singh is a
[2] "Mr.  Venkat is a
[3] "Dr. Jha is a good
>
> paste(names, "is",
[1] "Prof. Singh is a                                Jha is a good p$
> |
```

Rgui Configuration Editor

Single or multiple: [✓ MDI] SDI; [✓ MDI toolbar]; MDI statusbar
Pager style: [✓ multiple windows] single window; Language for menus
Font: Courier New; [✓ TrueType only]; size; style: bold
Console rows: 20; columns: 95; Initial left: -2; top: -5
[✓ set options(width) on resize]
[✓ buffer console by default?]
buffer: 250000; lines: 8000
Cursor blink: Partial
Pager rows: 25; columns: 80
Graphics windows: initial left: -25; top: 0
Console and Pager Colours: background: wheat3; normaltext: wheat4; usertext: white

(Refer Slide Time: 18:17)



Operations with Strings

Example:

```
> paste(1, " is first", 2, " is second", 3, " is third",
sep = "#")
[1] "1# is first#2# is second#3# is third"
```

Observe the role of blank space inside double quotes

```
> paste(1, "is first    ", 2, "is second    ", 3, "is
third    ", sep = "#")
[1] "1# is first    #2# is second    #3# is third    "
```

```
> paste(1, " is first", 2, " is second", 3, " is third",sep = "#")
[1] "1# is first#2# is second#3# is third"
>
> paste(1, " is first    ", 2, " is second     ", 3, " is third    ",sep = "#")
[1] "1# is first    #2# is second     #3# is third    "
> |
```

So, I have shown you it on the screen short also, right ok. Now, after this I try to take here some more examples so that you can understand the application of this paste. So, if you try to see here, I try to write down here paste then I am writing here two things, one is number and say another is character.

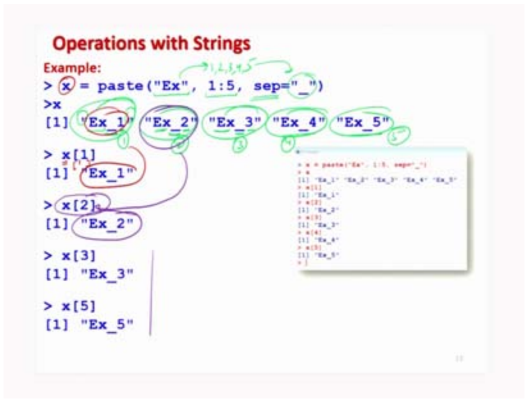
So, I am writing number character and then once again number and character and I try to repeat this 3 times. So, I try to use here number 1 and then the character is 1st, then number two and then within double quotes a character is 2nd and then number 3 and then a character within double quotes is 3rd. And I want to separate them using the hash sign.

So, now if you try to see if you try to see the outcome it will look like this, that one hash because hash will come here, hash will come here, hash will come here, hash will come here, hash will come here, you can see here 1 hash, is 1st hash, 2 hash is 2nd hash, 2 hash is 3rd, right. And in case if you just modify this example and I try to give here some blank space within this double quotes, that is 1st is 2nd and is 3rd which are the character.

So, you can see here I am trying to add here this blank space, right, you can see here. Yes, same blank space and you can see here like this. And now you see the effect on the outcome and the separated is the same hash. You can see here this outcome here. Now, there is a space here also, right and then there is a space here also.

So, now, you can see here that what I am trying to see here that when you are trying to use the separator, then this blank space is also playing a role and if you want to give more space artificially then you can create such a thing without any problem, ok.

(Refer Slide Time: 19:48)



Operations with Strings

Example:

```
> x = paste("Ex", 1:5, sep="_")
>x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"

> x[1]
[1] "Ex_1"

> x[2]
[1] "Ex_2"

> x[3]
[1] "Ex_3"

> x[5]
[1] "Ex_5"
```

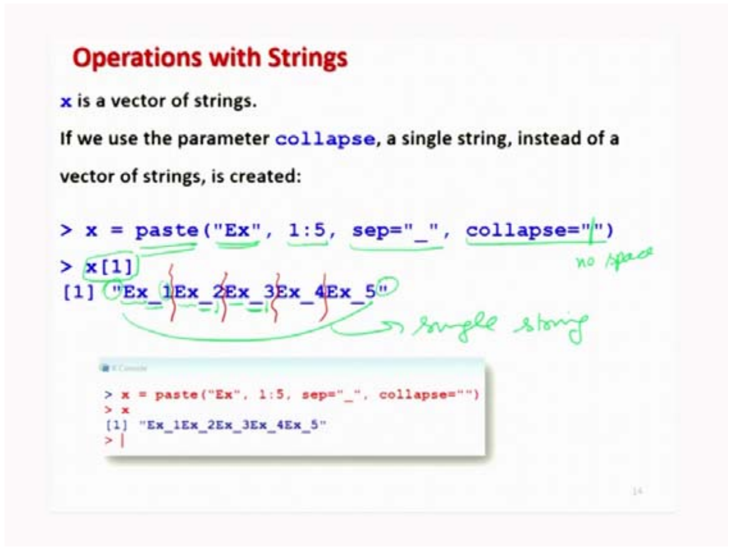
Now, after this I take one more interesting example, you can see here I try to take here a character here within double quotes Ex we could not call it as a short form exercise. And then I am trying to call here the numbers 1 colon 5. So, this is 1, 2, 3, 4 and 5 and after that I write down here the separator sep is equal to say underscore within the double quotes.

Now, if you try to see what will be the outcome this x is going to be attached, this Ex is going to be attached with each of this number and then these numbers are going to be separated by this underscore sign and this will create one say string for each of the number. So, you can see here the outcome looks here like this x, then the separator underscore and then the number here 1, then Ex then separator and then number 2 then Ex and then underscore number 3 and the same thing is with here 4 and 5.

And each of them is a separate string, you can see here they are in double quotes. So, 1st string, 2nd string, 3rd string, 4th string, 5th string. So, you can call it number 1, number 2, number 3, number 4 and number 5. Now, suppose I want to call here a particular string. So, how to call it? So, I am trying to store all these outcomes in a variable here x. So, suppose I want to call the first strain.

So, what I have to do here? That I simply have to write down here x and then within square bracket the index number. So, I have to write down here 1 and it will give me here this value here you can see. And similarly, if I want to call here the second value in the x, I have to simply write down here x and inside the square bracket two which is giving the location of the second string or the index and it will give you here the second value in the outcome Ex 2 and similarly you can go for say x 3, x 5, etc. So, that is not very difficult thing, right.

(Refer Slide Time: 21:37)



Operations with Strings

x is a vector of strings.

If we use the parameter collapse, a single string, instead of a vector of strings, is created:

```
> x = paste("Ex", 1:5, sep="_", collapse="")
> x[1]
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```

no space
single string

```
> x = paste("Ex", 1:5, sep="_", collapse="")
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
>
```

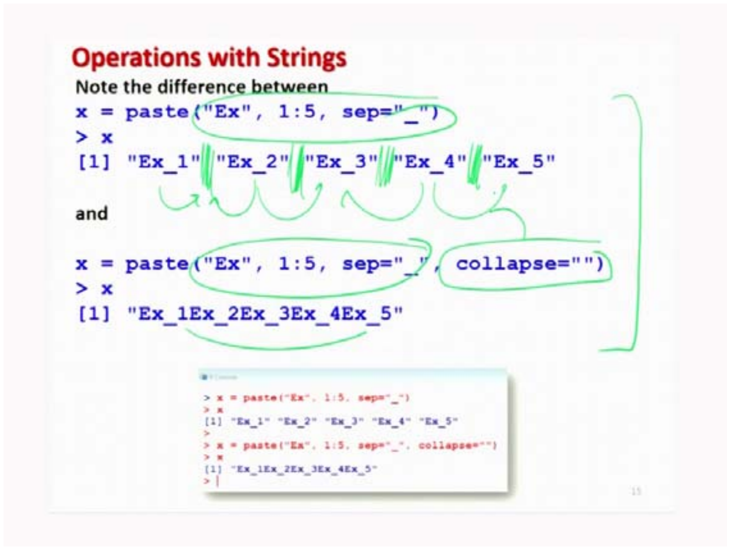
And now in the same line I try to consider the same example, but I try to use here the collapse also. So, what happened that I take the same example paste, then within double quotes Ex and the number one to 5 and the separator here is underscore that is the same thing which we considered in this earlier example here you can see here.

But now I add here collapse is equal to now, there is no space here. It is just two double quotes, right. Now, if you try to see what happens, whatever was the outcome earlier that there was some space here which I am trying to now mark in the red color, this space this is space, this is space, this is space.

This is now removed and now you can see here, there is no space here, no space here, no space here and no space here, but all the other operations are going to be the same is trying to take here Ex then it is trying to separate the other value by the underscore and then it is trying to take it here 1. And similarly, then it try to take here Ex then underscore and then 2, then Ex underscore and then 3 and so on.

But now there are only two double quotes. So, this is a single string, right. So, here if you want to call the first string, then it will you can also write down here as x within square bracket as 1, but it will give you the same outcome. There will not be x 2 or x 3 or like this, right.

(Refer Slide Time: 22:57)



Operations with Strings

Note the difference between

```
x = paste("Ex", 1:5, sep="_")
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
```

and

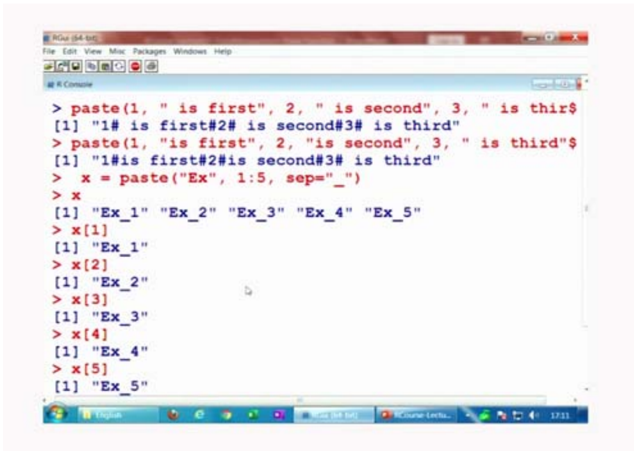
```
x = paste("Ex", 1:5, sep="_", collapse="")
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```

```
> x = paste("Ex", 1:5, sep="_")
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
>
> x = paste("Ex", 1:5, sep="_", collapse="")
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
> |
```

So, and if you try to see the difference between the two outcomes, I have put both the things on a single screen. So, you can see here this part is the same in both the cases, only this collapse is changing, but now because of this whatever was the space between the two numbers between the two strings here in the 1st case, which is this is removed in the 2nd case, right they are joined together and that is the role of the collapse set.

All this exercise 1 and exercise 2, exercise 3 and exercise 4 and exercise 5, they just collapse together and they are joined, right. So, let me try to show you first these examples and then I try to give you one more, result with paste and paste 0.

(Refer Slide Time: 23:45)

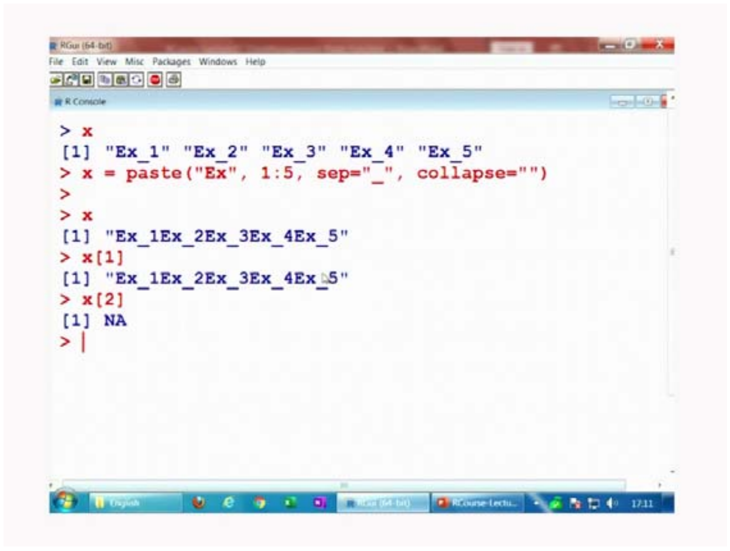


```
> paste(1, " is first", 2, " is second", 3, " is thir$
[1] "1# is first#2# is second#3# is third"
> paste(1, "is first", 2, "is second", 3, " is third"$
[1] "1#is first#2#is second#3# is third"
>  x = paste("Ex", 1:5, sep="_")
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
> x[1]
[1] "Ex_1"
> x[2]
[1] "Ex_2"
> x[3]
[1] "Ex_3"
> x[4]
[1] "Ex_4"
> x[5]
[1] "Ex_5"
```

So, if I try to take here this type of operation here, you can see here like this. So, you can see here this is the one is first etc. and suppose I just remove the space and that is all and then you try to compare. You can see here one is there is 1 space here and there is 1 is there is no space here. And the same thing can happen together also, right. So, similarly if you try to take here this example here, where you are trying to paste exercise one to 5 with the separator underscore.

You can see here x comes here like this and if you want to call here x the first value then you write x and 1 within the square bracket there will be this will give you the 1st value. If you want to call the 2nd value just write x 2 inside the square bracket and for the 3rd value just write 3, for the 4th value just write 4, for 5th value you write only Ex 5, right.

(Refer Slide Time: 24:40)

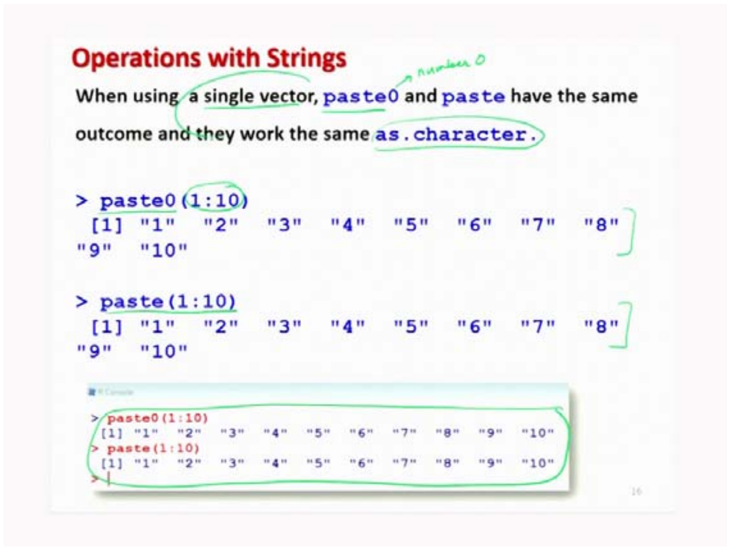


```
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
> x = paste("Ex", 1:5, sep="_", collapse="")
>
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
> x[1]
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
> x[2]
[1] NA
> |
```

So, this is how you can do it very easily. Now, in case if you try to use here the parameter collapse also, then what will happen? So, you can see here this is your here now you are trying to use a like this x is equal to paste, with the same up to here this is the same, but now here I have collapse. So, you can see here this comes out to here like this. So, this separation, this is removed by this collapse operator and you get here only here a single string.

For example, now if you want to call the first value in this new string x, this is like this. But if you try to call the second string, there is no because there is only 1 because all of them are joined into a single string, right, ok.

(Refer Slide Time: 25:12)



Operations with Strings

When using a single vector, paste0 and paste have the same outcome and they work the same as.character.

```
> paste0(1:10)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"
"9"  "10"

> paste(1:10)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"
"9"  "10"
```

```
> paste0(1:10)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
> paste(1:10)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
>
```

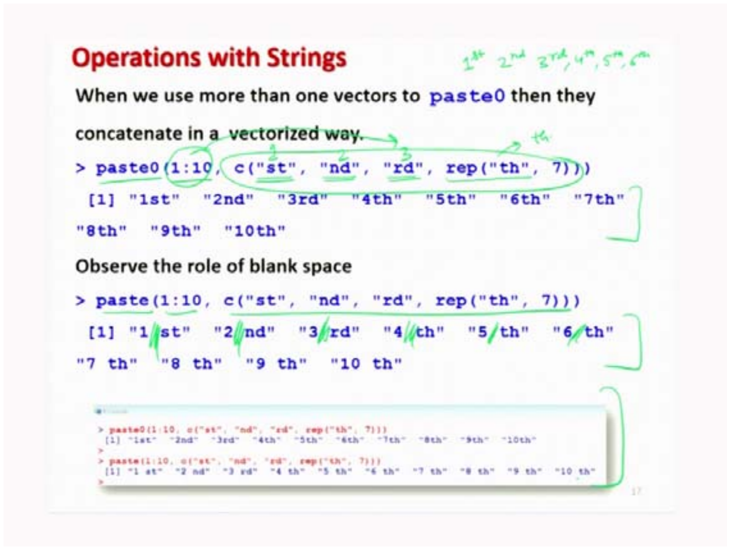
16

So, after this I give you very interesting example for the use of paste and paste0, right. So, when you are trying to use the paste function there is an alternative which is paste0, which is written here like the paste and then this is here number 0, right. So, actually both of them they have got the same outcome when they are used over a single vector and they were thus exactly in the same way for example, in the paste I have shown you that first all the values inside the argument they are converted using the command as dot character.

And then they are joined together the same thing happens in the paste0 also. For example, if you try to write down here a single vector say one to 10 which are the values one to 10 and if you want to use here paste or paste 0, you can see here when you use paste0, this has this type of outcome.

And when you try to use here paste command over one colon 10 then they have the same command actually and if you try to see there is no difference that you can observe very clearly from the screenshot. There is no difference in the use of paste0 and paste when you are trying to work with single vector for the values 1, 2, 3, 4 up to 10.

(Refer Slide Time: 26:17)



Operations with Strings

When we use more than one vectors to `paste0` then they concatenate in a vectorized way.

```
> paste0(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"
"8th"  "9th"  "10th"
```

Observe the role of blank space

```
> paste(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1 st"  "2 nd"  "3 rd"  "4 th"  "5 th"  "6 th"
"7 th"  "8 th"  "9 th"  "10 th"
```

```
> paste0(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"  "8th"  "9th"  "10th"
>
> paste(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1 st"  "2 nd"  "3 rd"  "4 th"  "5 th"  "6 th"  "7 th"  "8 th"  "9 th"  "10 th"
>
```

But now I try to show you under what type of condition they are going to make a difference, right. Suppose I try to take here the numbers here paste0, with number 1 to 10 and after that I want to concatenate them in a vectorized way and for that I try to give here a data vector here, which has like within double quotes “st” then within double quotes “nd” and within double quotes this is “rd”.

What are these things? You know, like when we write the 1st so we write one like “st”. When we write 2 so this is your “nd” when we write third it like a three and then “rd” and so on. So, these are those “st” “nd” and “rd” and then I try to repeat here see here this th for 7 time, right. For example, if you try to say fourth and then here 5th and then 6th and so on, 7th, 8th, 9, 10th etcetera, right.

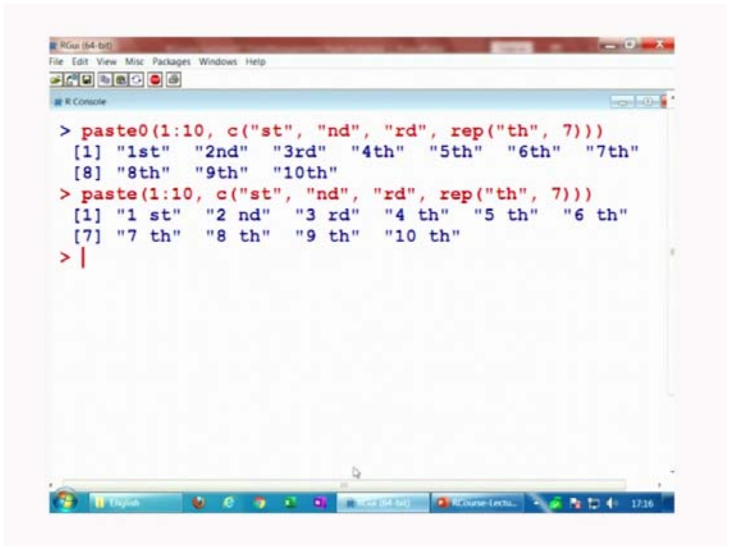
So, this is what I want to print. So, if you try to see now, I have made it clear what I want. II want to print here 1st, 2nd, 3rd, 4, 5th, 6th, 7th, 8th, and 9th, and 10th. So, for that I am trying to get the numbers 1, 2, 3, 4 from this statement and after that these alphabet like as “st” “nd” etcetera they are coming from another vector, which is here like this. We can see here now the outcome is going to be here like this, 1st the number 1 will be joined with this, here first number 2 will be joined here with the 2nd.

Then 3rd 3 will be joined with here the 3rd from this vector and then 4th 5th, 6th, 7th, 8th, 9th, 10th which are total 7 they will be joined here with the “th”. So, this you can see here that how efficiently you can operate with this thing, with paste0 command. Now, the question is cant you do the same thing with the paste command also? Yes you can do it, but if you try to see the difference in the outcome, it will be clear. So, you can see here paste 1 to 10 and the same command here.

I simply try to replace paste0, with paste and you can see here this is the outcome, there is a blank space which is given by the paste command which is the defaults single space. Unless until you try to write down here the collapse, you remove this space this will not work. So, this is the only advantage of using the paste0 command and this is here the screenshot.

So, let me try to show you these outcomes on the R console here and then I will try to then we will finish this lecture today.

(Refer Slide Time: 28:28)



```
> paste0(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"
 [8] "8th"  "9th"  "10th"
> paste(1:10, c("st", "nd", "rd", rep("th", 7)))
 [1] "1 st"  "2 nd"  "3 rd"  "4 th"  "5 th"  "6 th"
 [7] "7 th"  "8 th"  "9 th"  "10 th"
> 
```

So, you can see here this is here like this, this is like this and if you try to simply remove here paste0, with paste like this it is like this. So, this is how we can actually work with paste and paste0 and now we come to an end to this lecture and you have seen that this was a pretty interesting lecture that using the paste command and if you try to think in a different way you can do wonders, right.

So, its not actually wonder these are the needs, these are the requirements because when you try to produce a report, then definitely every report has some specific requirement.

And in order to fulfill those requirements you have two options either you try to write down a completely new function. Or you try to use these commands intelligently and try to combine them together so that they give you the same output which you would obtain after writing a new function.

I am not doubting on your capability to write a new function, but this is much straight forward and easier. And since this is an elementary level course. So, I would not recommend you at this moment to write the function to do these things which can be obtained directly by using these commands. And that was the advantage of the R software. So, now, once again you have ample of opportunities to create the examples, try to think what you can do, what are the different possibilities with the R software.

So, why do not you take some more examples and try to see that wherever you are working, what is the whatever is your need, whatever type of report. You always generate try to take some segments of that report and try to see how you can do all those things with the R software. So, with this request, I stop here and I will see you in the next lecture, till then goodbye.