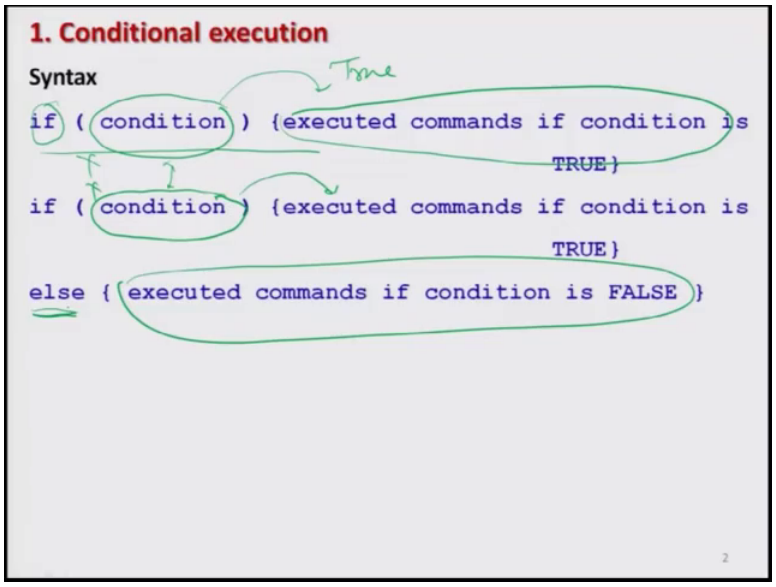
Introduction to R Software
Prof. Shalabh
Department of Mathematics and Statistics
Indian Institute of Technology, Kanpur

Lecture - 12
Conditional Executions and Loops

Welcome to the next lecture on the course Introduction to R Software. You may kindly recall that in the earlier lecture, we started the topic of conditional execution. And we had seen that there are different ways to get it done. And out of those ways we had discuss only the one option in the last lecture.

(Refer Slide Time: 00:42)



1. Conditional execution

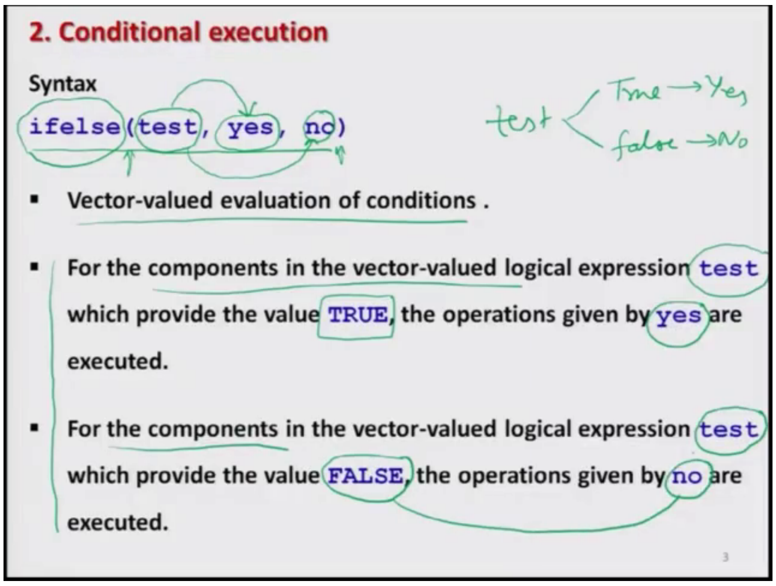
Syntax

```
if ( condition ) {executed commands if condition is
                                        TRUE}
if ( condition ) {executed commands if condition is
                                        TRUE}
else { executed commands if condition is FALSE }
```

And just for a quick revision on the last lecture we had done the, if conditional execution and we have taken different example by which we try to learn; that how the if statement can be executed. The rule was very very simple, you have to simply write here if, and then whatever condition you want to check here, inside the bracket and if this condition is true: then all this statement whatever you are writing here this will be executed, and if this condition is not satisfied, then you can put here another condition, if this condition is true, then this will be executed.

And suppose, if both of these conditions are not true, then the third condition which is written here that will be executed, but this condition is followed by the statement else, this is called as if else statement.

(Refer Slide Time: 01:44)


2. Conditional execution

Syntax

```
ifelse(test, yes, no)
```

- Vector-valued evaluation of conditions .
- For the components in the vector-valued logical expression `test` which provide the value `TRUE`, the operations given by `yes` are executed.
- For the components in the vector-valued logical expression `test` which provide the value `FALSE`, the operations given by `no` are executed.

test
True → Yes
false → No

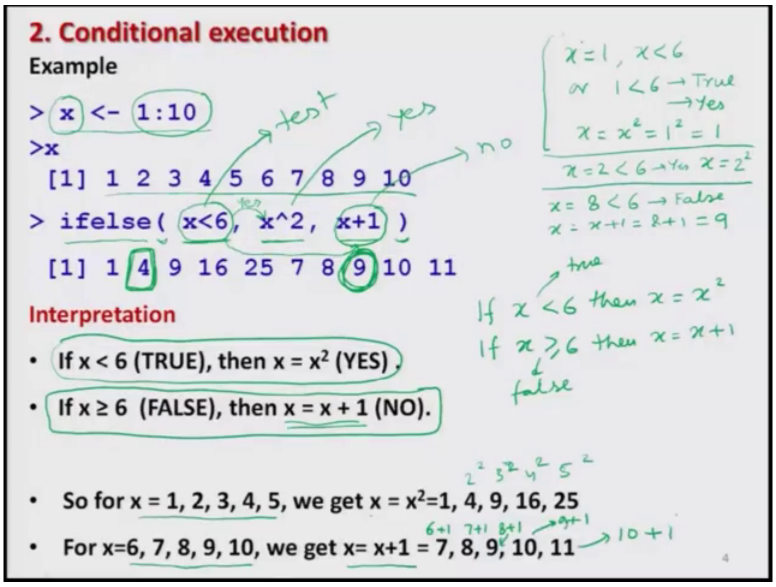
Now after this, let us try to look into the another statement. This is, a sort of if else statement, but this has a different format, and this has different types of utilities. First of all, if you try to understand what is the syntax? The part which I am circling this is if else i f e l s e all in small letters. After this inside this bracket, I am writing here test. This is a statement which you want to test.

Now, when you are testing a statement, there are two options. This statement can be true or this statement can be false; that means, the answer will be either yes, when the statement is true or the answer will be no if the statement is false. So, in case if the statement is true: then the control comes over here and here, whatever I have written as say y e s yes, here I have to write down the statement which has to be executed. Next option is that: suppose this condition which I am trying to test this is false in that case the control comes over here, what I have written here as say no and then whatever the statement I have written in place of no that is executed.

So, you can see that under the same format, this syntax is giving us an opportunity to write down the condition and based on that if the condition is correct or not, the appropriate statement is executed. One advantage of a this statement is this that a this statement can be used over the vector valued evaluations and the conditions can be checked. And here in this part you can read it I am trying to explain, what do you mean by here true and false. I am simply trying to see here that when I am trying to take a vector valued expression then the components in the vector valued expressions are logically tested by the expression what is mentioned under the test, and in case if the logical answer of this statement comes out to be true, then whatever a statement I have written under the yes, that is executed.

And similarly, the opposite also holds true. That whenever I am trying to consider any component of this vector; then whatever is the statement given under the test, is executed and if the answer is false: then whatever a statement I have given under here no, that is executed right ok.

(Refer Slide Time: 05:08)



2. Conditional execution

Example

```
> x <- 1:10
>x
 [1] 1 2 3 4 5 6 7 8 9 10
> ifelse( x<6, x^2, x+1 )
 [1] 1 4 9 16 25 7 8 9 10 11
```

Interpretation

- If $x < 6$ (TRUE), then $x = x^2$ (YES)
- If $x \geq 6$ (FALSE), then $x = x + 1$ (NO).

- So for $x = 1, 2, 3, 4, 5$, we get $x = x^2$=1, 4, 9, 16, 25
- For x=6, 7, 8, 9, 10, we get x= x+1 = 7, 8, 9, 10, 11

test
yes
no
$x=1,\ x<6$
or $1<6 \to$ True $\to$ Yes
$x = x^2 = 1^2 = 1$
$x=2<6 \to$ Yes $x=2^2$
$x = 8 < 6 \to$ False
$x = x+1 = 8+1 = 9$
true
If $x<6$ then $x = x^2$
If $x \geq 6$ then $x = x+1$
false
$2^2\ 3^2\ 4^2\ 5^2$
6+1 7+1 8+1 9+1
10+1

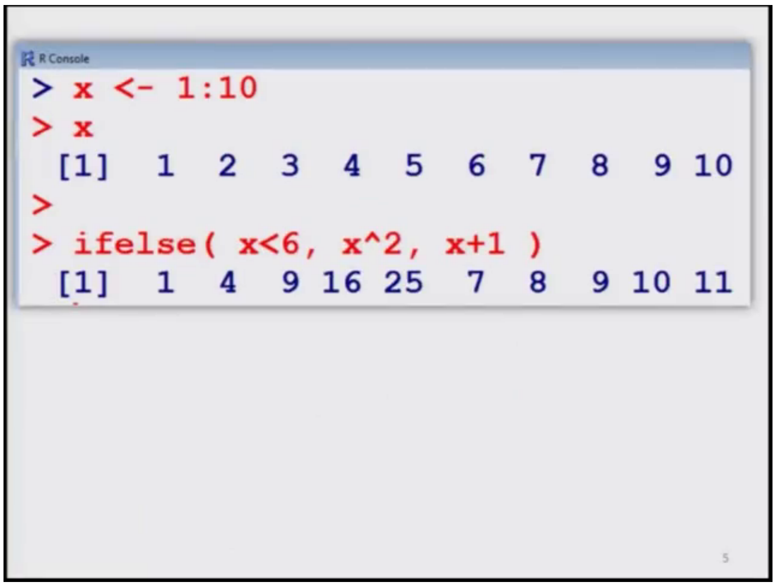
So, now let us try to take some examples and try to understand it. What do I really mean by saying these statements? Suppose I try to generate a data set, containing the value 1, 2, 3, 4 up to 10. And I try to store all this value under a variable, say here x. So, by this statement x less than hyphen 1 colon 10, this is a statement to generate a sequence of numbers starting from 1 to 10 at an interval of plus one. Well, you may get surprised at how this statement comes into picture.

But as I told you earlier, very soon we will be discussing all these types of different possibilities to generate the numbers or to generate the data set. Here you can believe on me that this will generate a sequence of numbers from 1 to 10. Now if you want to verify, what is the outcome you can see here that as soon as I try to type x on the R console, this gives me this output. Now I am going to use this data and I would like to expose it over the condition or the syntax if else. Now, what I want to do here if you try to see; first I am writing here if else, then inside these bracket signs I am writing here first statement x less than 6, second statement is x square and third statement is x plus 1.

Now, in case if you go by a rule, then earlier I had written here 3 things you can see here just for the sake of understanding, I am circling it test yes and no. Now here in this case, this becomes my test, this becomes my statement for yes and this becomes my statement for no. What does this mean? As we had discussed, the control will come over here the statement under test. And it will try to test whether this statement is right or wrong, true or false. In case if the statement is correct: then this is going to be executed. And suppose if this statement turns out to be false for a given value of x then the next statement x plus 1 will be executed.

So, in very simple words I have written here, but I will write down here that if x is less than 6 then the value x will be replaced by x square and if x is smaller than 6 is true, and if x is greater than or equal to 6 is false. So, this becomes my true value, and this becomes my here false value. And in this case, x is going to be replaced by the value x plus 1. Before I go further, let us try to execute it see the outcome and then we will try to understand how the things are happening. First I try to generate the data here over the R console. Now, you can see here that this is giving me a sequence of values from 1 to 10.

(Refer Slide Time: 08:44)



```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> ifelse( x<6, x^2, x+1 )
 [1]  1  4  9 16 25  7  8  9 10 11
```

Now, I try to write down this condition over here. So, you can see here, this is the outcome which I am getting. The screenshot of this statement is given in the next slide that you can see this is the same thing what we are observing, from the R console. Now let us come back on the earlier slide, and try to see, what does this R is trying to do. Now, let us try to see what really this program is doing. You can see here x takes value 1 to 10. So, let us try to take here first value x equal to 1.

Now, you can see here I have to compare with the condition x less than 6; that means, 1 is less than 6. There are 2 options. Either this can be true or this can be false. What do you think? Yeah; obviously, 1 is a smaller than 6, so this condition is true; and so this becomes my here yes. And when this condition is true you have seen here then the condition becomes here that value of x is going to be replaced by x square and this becomes 1 square and this is equal to here 1. Now, since we have mentioned that in this case, the control goes over the entire vector.

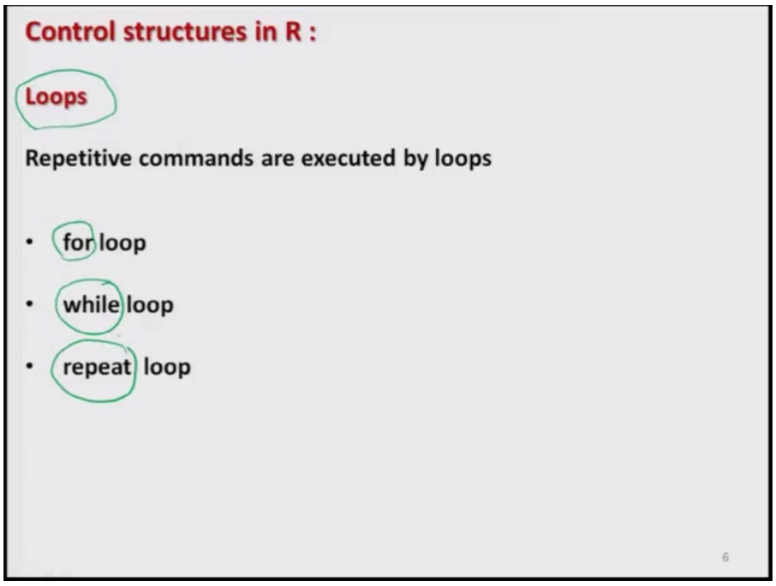
So, here I have taken the first value of the vector. Now the control will go to the second value. What is second value? X equal to 2. Now again I have to check whether this value is smaller than 6 or greater than 6; it is smaller than 6, answer again comes out to be yes and then x will be replaced by x square which is your here 2 square that is, 4; and that is why you get here this outcome. And similarly I try to take here another example say here x equal to suppose say here 8. Now the condition under the test will check whether 8 is smaller than 6. The answer is no, this is not correct.

So, the answer is that this statement is false. And when this statement is false; then we can see here, that the second condition under no, that has to be executed. And in this case we can see here, in this case x is going to be replaced by x plus 1, and this will become here 8 plus 1 which is equal to here 9. And you can see here, this is the value here which is here 9 and similarly the entire expression will be executed. And all these expressions I have given here that for x equal to 1, 2, 3, 4, 5 the conditions remain true, that each of this value is smaller than 6, so the variable x is replaced by x square that is 1, 2 square, 3 square, 4 square and 5 square.

And as soon as, the condition is violated; that means, for the numbers 6, 7, 8, 9 and 10. The condition that x less than 6 is not true, but it is false. So, in this case the variable is replaced by x plus 1. Now, in this case the values will be replaced by 6 plus 1, 7 plus 1 which is 8; 8 plus 1 which is here 9, 9 plus 1 and here 10 plus 1. So, now, you can see based on this conditional statement, whether the condition is true or not the required statement is executed and the values move in a vector.

After, this let us try to take a new topic which is again a control structure in R.

(Refer Slide Time: 13:35)



Control structures in R :

Loops

Repetitive commands are executed by loops

- for loop
- while loop
- repeat loop

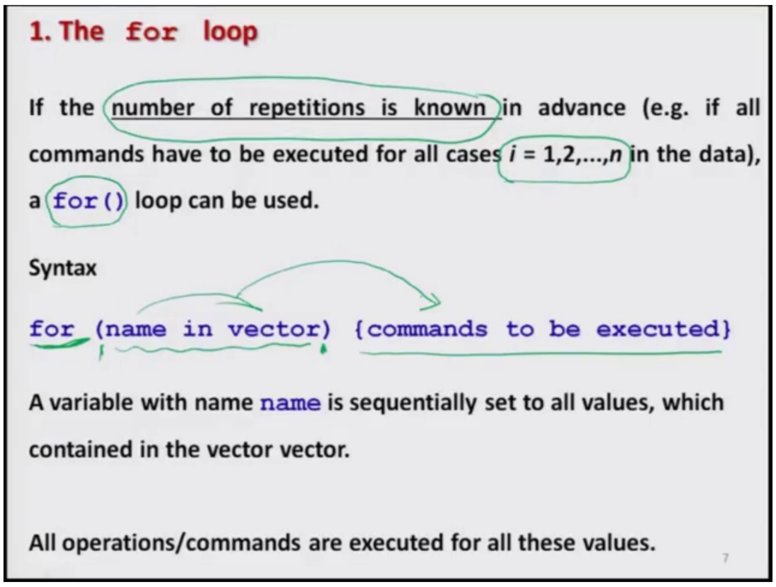
6

And, this is about loops. What is the loop and what is its meaning in the programming language. Whenever, we are trying to write a program, many times we have a requirement that some statements have to be repeated for certain number of times. The number of times can either be known, or they may be unknown to us. In case if the number of times, a command has to be repeated is known to us that means, the same process has to be repeated again and again inside a loop.

The second situation can be that we really do not know that how many times a process has to be repeated, but we have certain conditions. For example, a student can be asked to appear in an exam unless and until; he gets 70 percent of marks. So, we really do not know whether the 70 percent marks will come in the first exam or the second time or the third time and or say so on. Second situation can be a student knows that I have to appear in 3 exams. So in the second case, the number of repetitions are fixed, but in the first case the number of repetitions are unknown to us, and they will be known to us only when the condition is satisfied that means, the student gets the required marks right.

So, in order to do such a repetitive a process of commands or the execution of repetitive commands, we are going to discuss here 3 types of loop. One is called for loop, other is called while loop, and third is called repeat loop. And these three loops are used under different types of condition you will try to understand their syntax as well as their examples.

(Refer Slide Time: 15:51)



1. The `for` loop

If the number of repetitions is known in advance (e.g. if all commands have to be executed for all cases $i = 1,2,...,n$ in the data), a `for()` loop can be used.

Syntax

```
for (name in vector) {commands to be executed}
```

A variable with name `name` is sequentially set to all values, which contained in the vector vector.

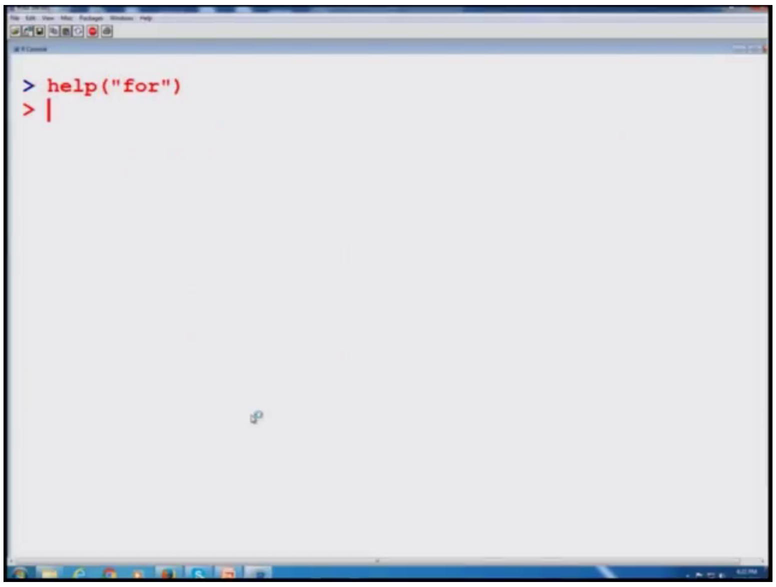
All operations/commands are executed for all these values.

First, we take up the; for loop. The for loop is useful, when the number of repetition is known to us; that means, before I am starting my programming, I know in advance that how many times the program has to be repeated.

For example, if I say that the process has to be repeated say small a number of times. First time, second time, third time up to nth time, where n is some finite number. In this situation it is recommended to use a for loop. And the syntax of for loop is very simple, just try to write down here for and inside these brackets try to write down the name of the vector in which the values are assigned and inside this curly bracket you have to write all the commands which have to be executed. And then whatever the values are contained in the vector over here, they are sequentially computed for the statements which are written inside the curly brackets and all operations are done whatever we have mentioned it.

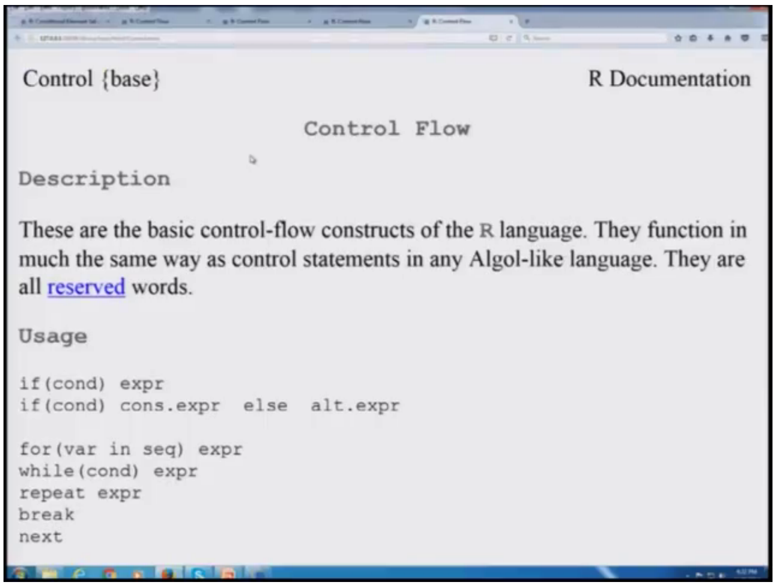
Now, the question comes, suppose I want to know more about this for. Then the rule is very very simple, just go to the R software and then try to get here help. For example, here I can type out here help and inside the inverted comma say here for and you will see here the control comes over here.

(Refer Slide Time: 17:40)



```
> help("for")
> |
```

(Refer Slide Time: 17:50)



Control {base} R Documentation

Control Flow

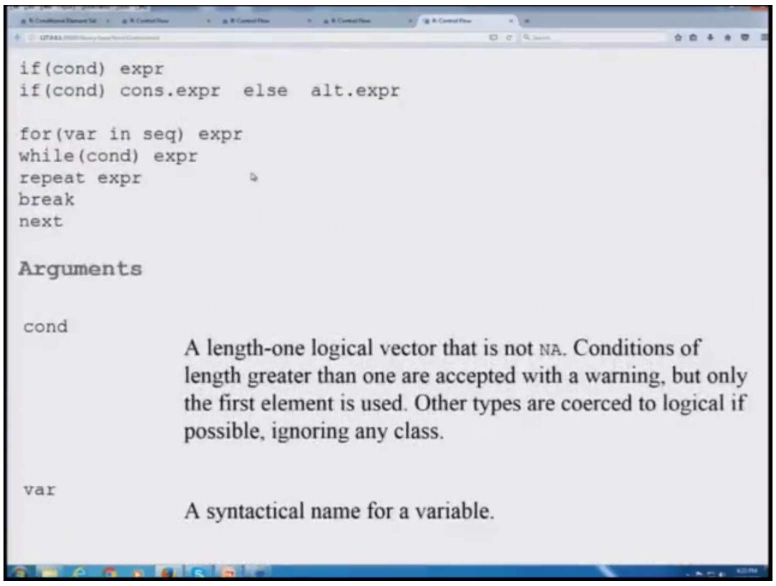
Description

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language. They are all reserved words.

Usage

```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

(Refer Slide Time: 17:59)



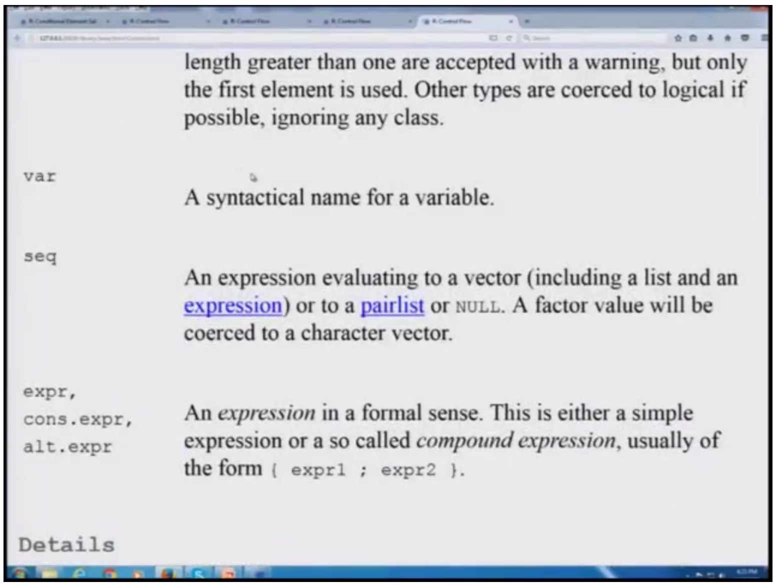
```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

Arguments

`cond`	A length-one logical vector that is not `NA`. Conditions of length greater than one are accepted with a warning, but only the first element is used. Other types are coerced to logical if possible, ignoring any class.
`var`	A syntactical name for a variable.

So, you can see here all these information has been, given online. And that is coming directly from the site of R project. And here they are trying to give all information about for loop, while loop, repeat loop and believe me this is the most authentic information.

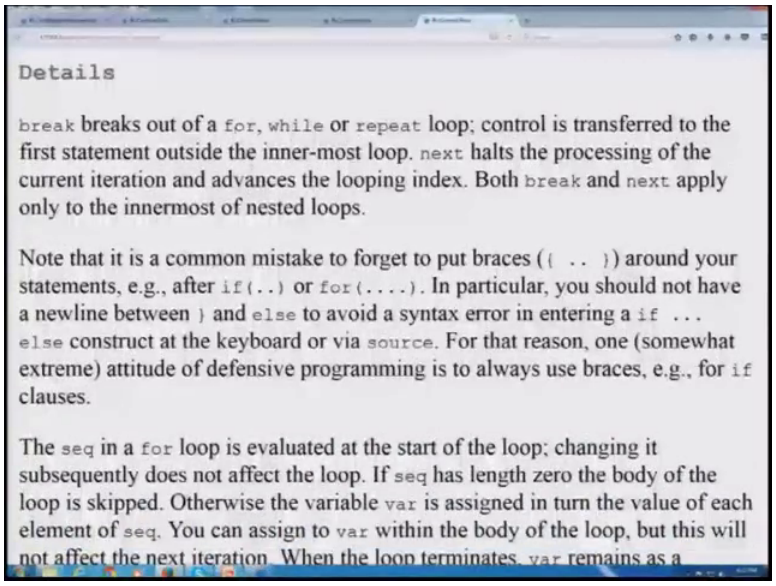
(Refer Slide Time: 18:04)



	length greater than one are accepted with a warning, but only the first element is used. Other types are coerced to logical if possible, ignoring any class.
`var`	A syntactical name for a variable.
`seq`	An expression evaluating to a vector (including a list and an expression) or to a pairlist or `NULL`. A factor value will be coerced to a character vector.
`expr`, `cons.expr`, `alt.expr`	An *expression* in a formal sense. This is either a simple expression or a so called *compound expression*, usually of the form `{ expr1 ; expr2 }`.

Details

(Refer Slide Time: 18:09)



Details

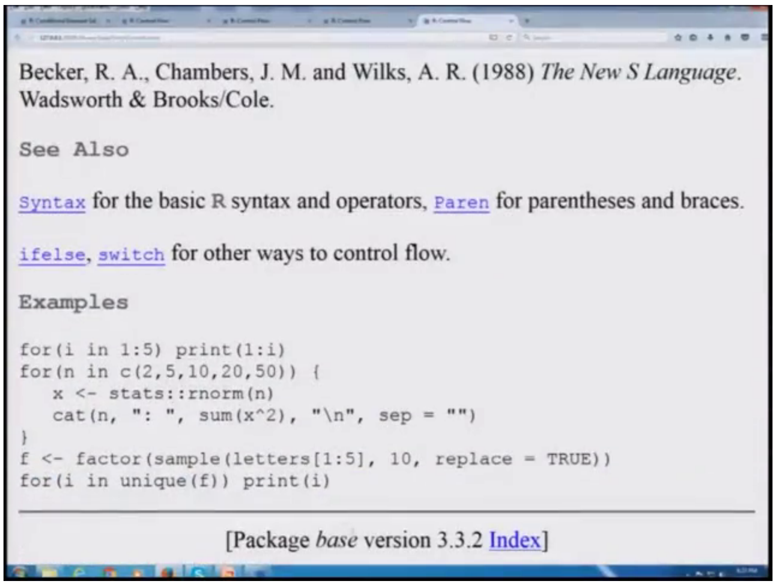
`break` breaks out of a `for`, `while` or `repeat` loop; control is transferred to the first statement outside the inner-most loop. `next` halts the processing of the current iteration and advances the looping index. Both `break` and `next` apply only to the innermost of nested loops.

Note that it is a common mistake to forget to put braces (`{ .. }`) around your statements, e.g., after `if(..)` or `for(....)`. In particular, you should not have a newline between `}` and `else` to avoid a syntax error in entering a `if ... else` construct at the keyboard or via `source`. For that reason, one (somewhat extreme) attitude of defensive programming is to always use braces, e.g., for `if` clauses.

The `seq` in a `for` loop is evaluated at the start of the loop; changing it subsequently does not affect the loop. If `seq` has length zero the body of the loop is skipped. Otherwise the variable `var` is assigned in turn the value of each element of `seq`. You can assign to `var` within the body of the loop, but this will not affect the next iteration. When the loop terminates, `var` remains as a

The only hurdle is that you need to read it carefully right.

(Refer Slide Time: 18:28)



Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`Syntax` for the basic R syntax and operators, `Paren` for parentheses and braces.

`ifelse`, `switch` for other ways to control flow.

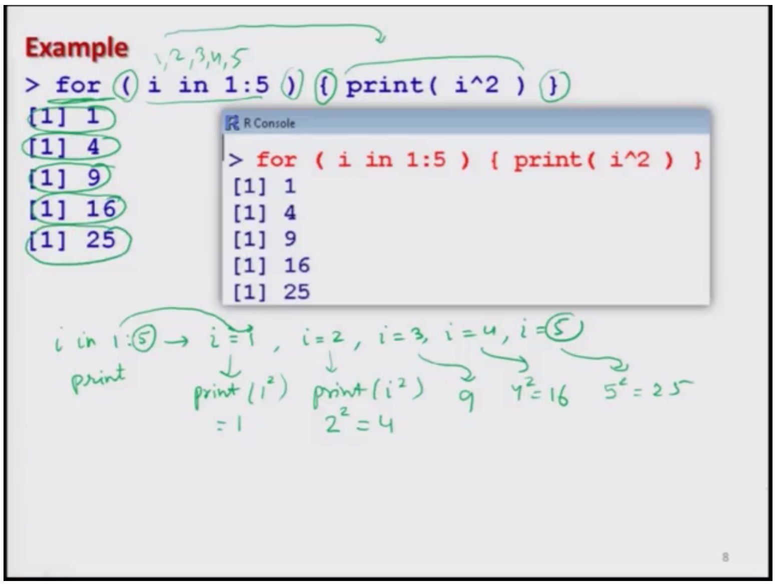
Examples

```
for(i in 1:5) print(1:i)
for(n in c(2,5,10,20,50)) {
   x <- stats::rnorm(n)
   cat(n, ": ", sum(x^2), "\n", sep = "")
}
f <- factor(sample(letters[1:5], 10, replace = TRUE))
for(i in unique(f)) print(i)
```

[Package *base* version 3.3.2 Index]

So, here we are trying to take, some of the things and then we are trying to understand it. Once you understand these basic functions, then depending on the need, you can execute these commands, you can read these syntaxes in more detail and can do the required job.

(Refer Slide Time: 18:49)



Example

```
> for ( i in 1:5 ) { print( i^2 ) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

R Console

```
> for ( i in 1:5 ) { print( i^2 ) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

Let us try to take a simple example and try to understand, how this for loop is executed. First if you try to understand what is written over here, first this is the statement for the for, then inside these 2 brackets, I am trying to write i in 1 to 5. This is a standard statement where I try to write i in 1 colon 5, this means the first value of i is going to be taken is i equal to 1, that is the starting value from say here.

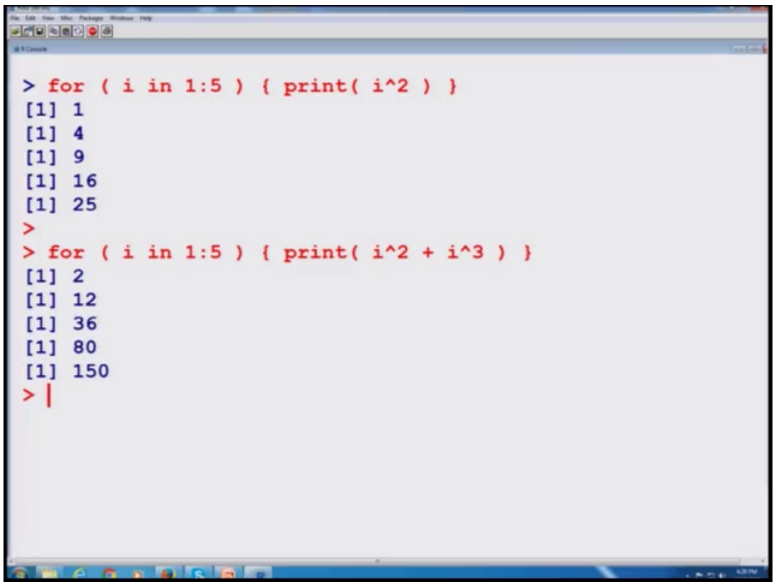
Then, the next value will be taken as here, i equal to 2. Next value will be i equal to 3, next value will be i equal to 4, and after this the last value will be i equal to 5, which is a here 5 is coming from here. This is the meaning of this. In case if you do not want to take these values in a sequence, then there is another command we can combine all the values inside the vector and you will see it in another example. Now in this curly brackets I am writing a very simple command, print i square. This print is a very simple command, to print the outcome on the screen.

So, what I am trying to say here that the values there are now 5 values here 1, 2, 3, 4 and 5 which are contained in this bracket. From here these values will be transferred to the statement inside the curly bracket one by one, and the statement will be executed and this outcome will be recorded for each and every value. Now, you see what will happen here. First of all i equal to 1 will go into the commands. And I am asking simply print i square; that means, 1 square. 1 square is 1. So, 1 has to be printed. Now in the next repetition this will take up the next value i equal to 2, and the command that print i square will be executed and the answer will be here 2 square which is equal to here 4. You can see here that here the answer is 1 and here the answer here is 4.

And, similarly this command will be repeated, and for i equal to 3 we will get the answer 9 that a 3 square, i equal to 4 i will get the answer 4 square which is 16, and for i equal to 5, we have an answer 5 square which is equal to here 25, and you can see here these values are printed here. So, you can see that the execution is happening one by one, one at a time, and for all the values in the vector. So just by doing or just by writing one simple syntax using the for loop, I can repeat the operation 5 times. And in case if I want to repeat it for more number of times, I simply have to change my index.

So, let us try to do it over the R console, and try to see what happens.

(Refer Slide Time: 22:38)

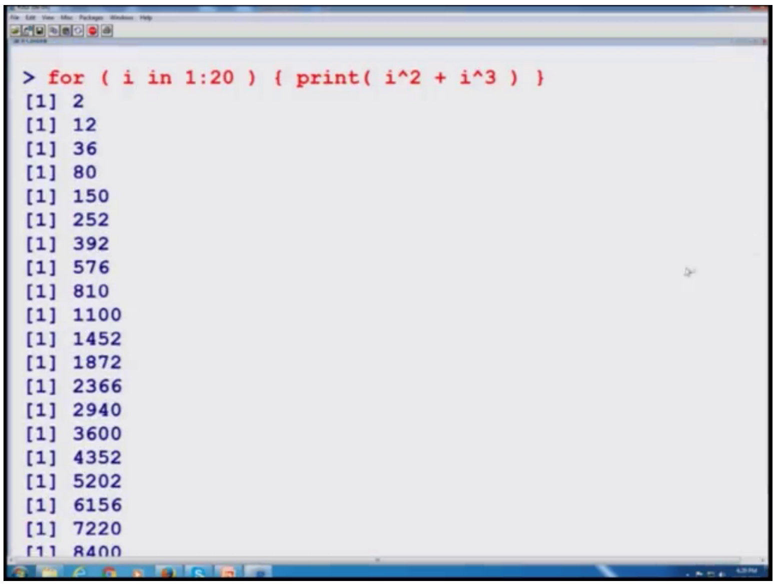


```
> for ( i in 1:5 ) { print( i^2 ) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
>
> for ( i in 1:5 ) { print( i^2 + i^3 ) }
[1] 2
[1] 12
[1] 36
[1] 80
[1] 150
>
```

You can see here, this is giving me 1, 4, 9, 16, 25, this is 1 square, 2 square, 3 square, 4 square, 5 square. Now suppose I want to get here i square plus say i cube. Now, I can do the same thing. Now you can see here whatever are the values are obtained, they are the values for i equal to 1, 2, 3, 4, 5, it is giving the me the value of 1 square plus 1 cube, 2 square plus 2 cube, 3 square plus 3 cube and so on.

And, suppose if I want to repeat this for process for say for more than say 5 times, Suppose, I say I want to repeat this process for say 20 times.

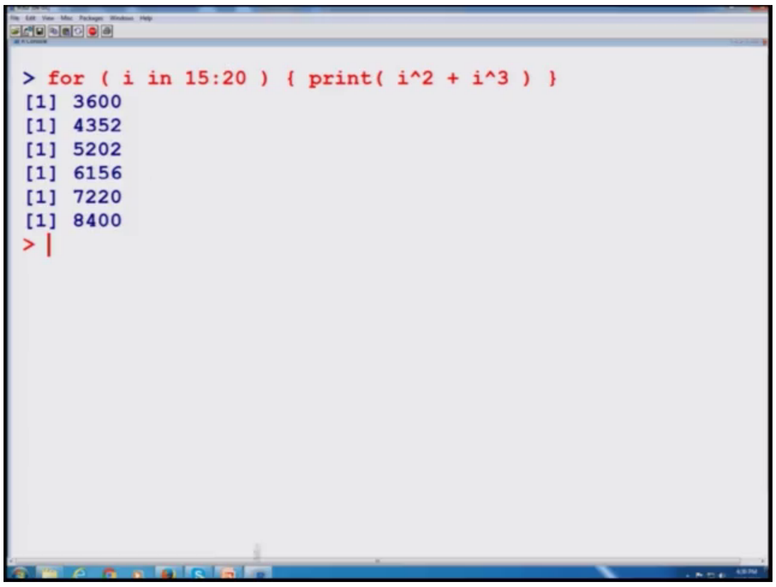
(Refer Slide Time: 23:26)



```
> for ( i in 1:20 ) { print( i^2 + i^3 ) }
[1] 2
[1] 12
[1] 36
[1] 80
[1] 150
[1] 252
[1] 392
[1] 576
[1] 810
[1] 1100
[1] 1452
[1] 1872
[1] 2366
[1] 2940
[1] 3600
[1] 4352
[1] 5202
[1] 6156
[1] 7220
[1] 8400
```

So, anyway no issues; just try to write down this command and try to change here the values say here 20. Now as soon as you press enter, you get here all the 20 values here.

(Refer Slide Time: 23:44)

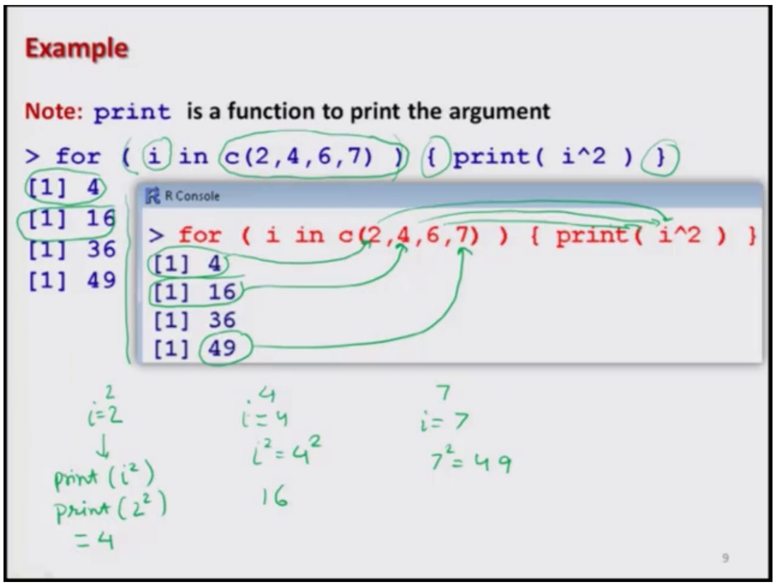


```
> for ( i in 15:20 ) { print( i^2 + i^3 ) }
[1] 3600
[1] 4352
[1] 5202
[1] 6156
[1] 7220
[1] 8400
> 
```

And, suppose if I say no I do not want to go by, from say 1 to 20, but I want to go only from say 15 to 20. Now the process will start from i equal to 15, and then this value will be calculated. So, you can see here, you have only here 6 values for i equal to 15, 16, 17, 18, 19 and 20.

So, you can see here that execution of this for loop is not difficult at all, but it is very very useful for us right.

(Refer Slide Time: 24:25)



Example

Note: `print` is a function to print the argument

```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }
[1] 4
[1] 16
[1] 36
[1] 49
```

R Console

```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }
[1] 4
[1] 16
[1] 36
[1] 49
```

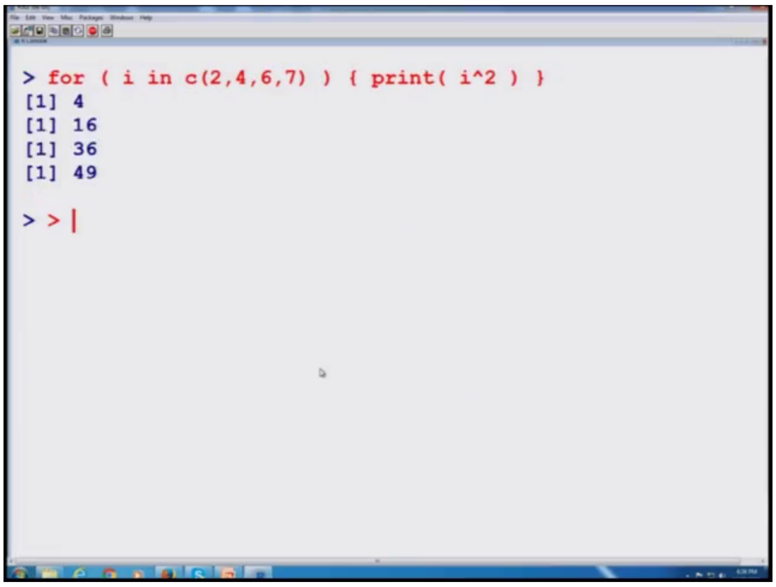
Now, let us try to take another example. In the earlier example, I have taken the values in a sequence. Something likes 1, 2, 3, 4, 5 and then 6, 7, 8, 9, 10. Now suppose you do not want all these values to be in a sequence, but you have some your own predefined values and you want to repeat the execution of command for those given values. So, just for the sake of simplicity, I have taken here the values 4 values, 2, 4, 6 and 7, and I have combined them inside a vector. And I am assigning it to a variable here i. Remember I will go sequentially one by one inside a vector, and that is the beauty of this for loop.

And, then these values will be taken from this statement inside this bracket. And then whatever is my statement inside this curly bracket that is going to be executed. In this case, also I try to use the same print command that is a function to print the argument right. So although, I have given you here the screenshot, but let us try to understand, what is really happening? So, you can see here that there are here 4 values, 2, 4, 6 and 7. Now, I takes here the first value, say here i equal to 2. And then the control goes to print i square, which is here print 2 square, and the answer comes out to be here 4. And you can see in the output either on the screen or over the screenshot that we are getting the same thing.

Now, I try to take some more values. I try to take here the next value which is here 4. So, I becomes here 4, and in this case the value of i square which is equal to here 4 square is printed and the answer comes out to be here 16. This is denoted over here, or the second value right. So, this thing correspond to i equal to 2, this correspond to i equal to 4, and similarly if you try to take the fourth value 7; that means, i equal to here 7 and the 7 square here is 49 and this is printed over here. So, you can see here that from this statement all these values are coming to this one, one by one.

And the commands are repeated for 4 times. So, if you have some other values you can again do it yourself and if you want to see its outcome, let us try to do it over the R console.

(Refer Slide Time: 27:38)



```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }
[1] 4
[1] 16
[1] 36
[1] 49

> > |
```

So, now you can see here the outcome, which is also given in the screenshot over here. Now, at this stage I stop here and I have given you one example of this loop. Now my request is that, you please try to practice it try to understand the concept of loop by taking several examples. And in the next turn I will try to take another 2 loops and we will try to move forward till then goodbye.