Control structures are important elements of any programming language and can be used to control the flow of code executions. This presentation provides a brief overview of some of the useful control structures in R and provides examples.

KENT STATE UNIVERSITY

Control Structures

- Control structures in R allow you to control the flow of execution of a series of R expressions.
- In other words, control structures allow you to put some "logic" into your R code, rather than just always executing the same code every time.
- Commonly used control structures are:
  - **if** and **else**: testing a condition and acting on it
  - **for**: execute a loop a fixed number of times
  - **while**: execute a loop while a condition is true

WWW.KENT.EDU

So why do we need control structures? Well, control structures allow you to put some "logic" into your R code, rather than just always executing the same R code every time. We will cover three important control structures in this presentation. If else condition statements, for loops and while loops.

KENT STATE UNIVERSITY

"if-else" statements

- The if-else combination is probably the most commonly used control structure in R (or perhaps any language).
- This structure allows you to test a condition and act on it depending on whether it's true or false.

```
## Generate a random number
x <- runif(1, 0, 10)
if(x > 3) {
y <- 10
} else {
y <- 0
}
x
## [1] 5.346585

Y
## [1] 10
```

WWW.KENT.EDU

So let's start with if-else statements; well if-else statements are probably the most commonly used control structures in R or perhaps in any other languages. This structure allows us to test the condition and to act depending on whether the conditions we are testing is true or false. Let's consider a simple example. On the right-hand side we can see that the variable x has been created as a random number between zero and ten. Then we define the value of another variable y based on the value of x. Such that if x is greater than three, we set the value of y to ten, or otherwise we set the value of y to zero. We'll then print x and y. in this example, x was randomly generated as 5.3, and therefore it was greater than three. As a result, the value of y was set to ten.

KENT STATE UNIVERSITY

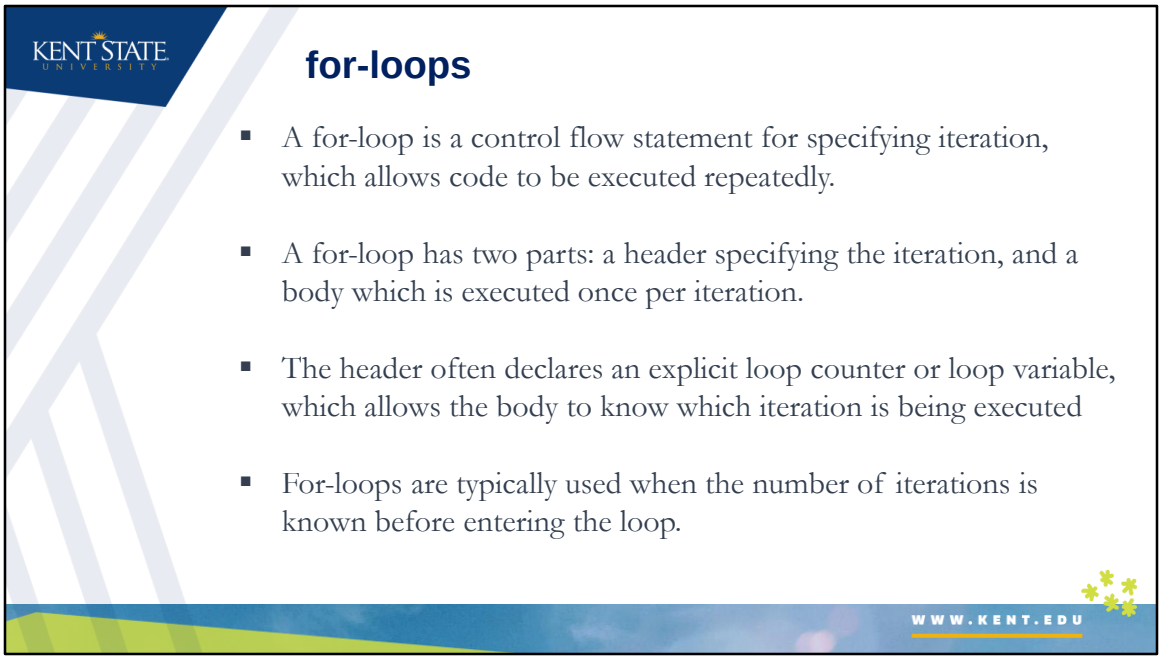
Example

- In the previous example, the value of y is set depending on whether x > 3 or not.
- This expression can also be written a different, but equivalent, way in R.

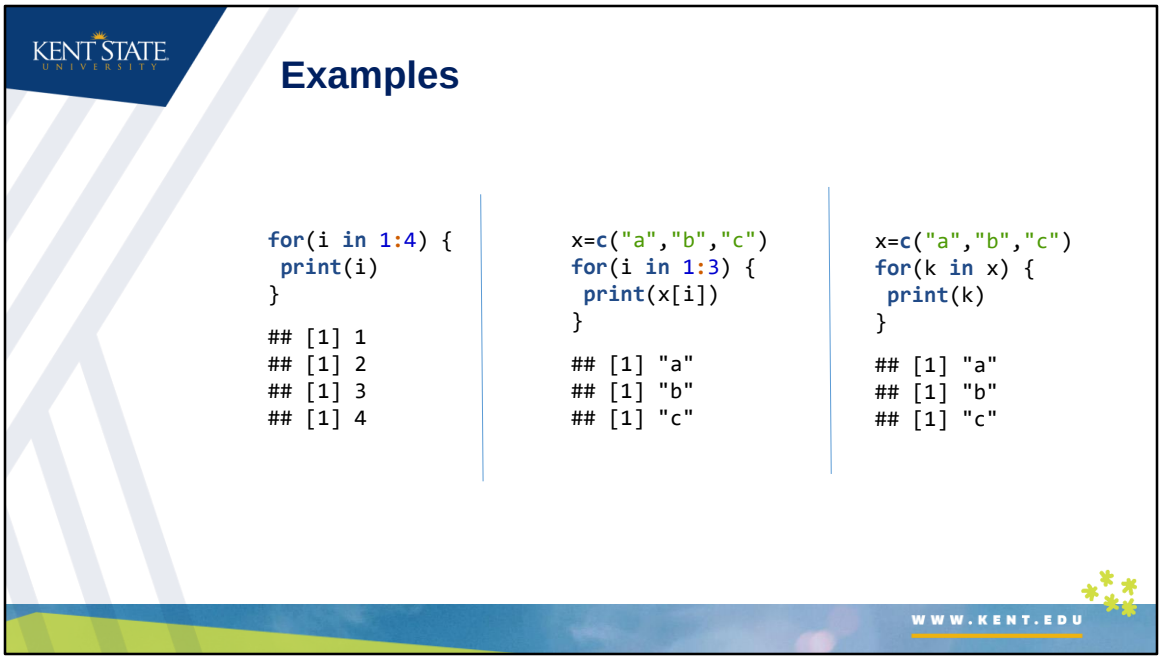
```
## Generate a random number
x <- runif(1, 0, 10)
y <- if(x > 3) {
10
} else {
0
}

x
## [1] 5.939933

Y
## [1] 10
```

WWW.KENT.EDU

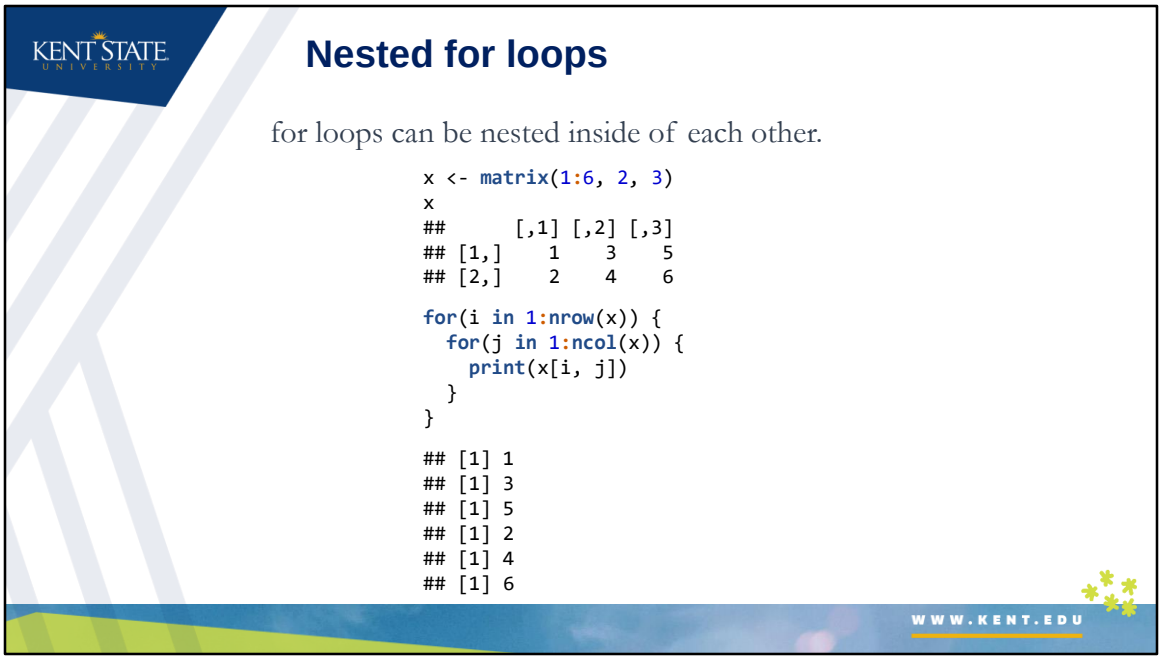
So in the previous example we saw that the value of y was set based on the value of x. More specifically if x was greater than three, the value of y was ten, otherwise it was zero. Well there's another way to represent the same logic. Again, we define x as a random number between zero and ten, and then we define y as if greater than three, and then we have brackets ten close bracket else zero. The result of the code is exactly the same. First the if condition has been tested, and depending whether the test condition is true or false, ten or zero would be respectively selected for y. The results are also shown in here. In this example, the random value of x was 5.9 and therefore y was set to ten.



For-loop is another useful statement to repeat a quote for a certain number of times. A for loop has two parts, a header which specifies the iteration, and a body of the quote which needs to be repeatedly executed. The header normally declares a loop counter or a loop variable, which allows the body to know which iteration is being executed. For-loops are generally used when we know the number if iteration beforehand.



Here are a few simple examples. So, in the first example we have for i in one to four, bracket print i closed bracket. So, this code is increasing the value of i from one to four and each time is printing the value of i. So, the output is one, two, three, four. Each time the loop is executed, the value of i is printed. In this case, i is the counter of the loop. We can also use the counter of the loop to get access to different elements of the vector. So, in this case we define x as a vector that contains a, b, and c. And then we define for i in one to three. We start the loop and then we print xi during each iteration. So, during the first iteration, the x1 which is a, is printed. During the second iteration, b, which is the second element of x is printed, and during the last iteration, the last value, or c is printed. The loop counter itself could be an element of a vector. So again, we can define x as a vector of a, b, and c, and so instead we can define k, the counter of the loop, as an element of x. So, in this case, we say for k, in x print k. So, in this example, k is going to be one element of x, during each iteration.



For loops can also be nested inside of each other. So here is another example, in here instead of a vector we have a matrix. The matrix has two rows and three columns. The first row is one three, five, and the second row is two, four, and six. Now we define two for loops where the second loop is nested inside the first loop. The counter for the first loop is i and the counter for the second loop is j. So, first we have for i in one to the number of rows for x, which in this case is two, and then we have the second loop which is for j in one to number of columns for x, which is three, print x, i, and j. So, during each iteration, elements i and j of x will be printed. So, as we can see during the first iteration, i is one and j is one so the value of one is printed. For the second iteration, i is still one, but j has increased to two, which means that the first row second column the value is three. So, three is printed and so on. Although nested loops can very useful, you should try to avoid having more than three levels of nesting inside a loop to make your quote more readable.

KENT STATE UNIVERSITY

while Loops

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement.
- The while construct consists of a block of code and a condition/expression.
- The condition/expression is evaluated, and if the condition/expression is true, the code within the block is executed.
- This repeats until the condition/expression becomes false.

WWW.KENT.EDU

A while loop is another control flow statement in R. A while loop can be considered as a combination of if else statement and a for loop. In a while loop, first a conditional expression is evaluated. If the result of the expression is true, the code within the loop is executed. The while loop continues to execute until the condition becomes false.

KENT STATE UNIVERSITY

Example

```
count <- 0
while(count < 7) {
  print(count)
  count <- count + 1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

WWW.KENT.EDU

Here's a simple example; in this case, we set the variable count to zero and so while count value is smaller than seven, we print the value of count. However, each time the loop is executed, we increase the value of count by one. So as we see, count is initially zero, the condition of the while loop is met and therefore the loop is executed. The count value is then increased by one, but the y condition still holds true. This continues until count is increased to seven. At that point the y condition is no longer valid and therefore the y loop is terminated.